UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Análisis de patrones de comportamiento en cursos online para la detección de fraude

Autor: Martínez Donoso, Blanca
Tutor: Pulido Cañabate, Estrella

Junio 2020

Análisis de patrones de comportamiento en cursos online para la detección de fraude

AUTOR: Martínez Donoso, Blanca
TUTOR: Pulido Cañabate, Estrella

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2020

Resumen

Este Trabajo Fin de Grado trata sobre la detección de usuarios fraudulentos dentro de un curso online de la plataforma edX, mediante el análisis de patrones de comportamiento. Para llevar a cabo este trabajo, hemos estudiado qué criterios aplicar para determinar si un usuario es sospechoso de haber cometido algún tipo de fraude, además de implementar el desarrollo técnico haciendo uso de la nueva tecnología, Big Data. Desarrollamos un proceso Spark que ha sido capaz de leer desde un directorio HDFS un fichero de logs generado por la plataforma y detectar comportamientos anómalos por parte de los estudiantes.

Asimismo, tras haber conseguido realizar una detección de usuarios fraudulentos, analizamos nuestros datos clasificados almacenados en un dataset y estudiamos la posibilidad de poder predecir este tipo de usuarios a través de sus interacciones con la plataforma, con el fin de ser capaces de detectarlos antes de que el curso se dé por finalizado. Para esto, hemos hecho uso de Machine Learning desarrollando un código en Python que entrena y predice con diferentes modelos de clasificación nuestro conjunto de datos, y comparamos los resultados obtenidos de cada uno calculando distintas métricas de evaluación, obteniendo finalmente cuál sería el modelo óptimo en nuestro caso en concreto.

Abstract

This TFG deals with detection of fraudulent users in online course from edX platform through analysis of behavioral patterns. We have studied which criteria to select for determining whether a user is suspect of cheating and we implemented the technical development using the new technology, Big Data. We developed a Spark process with which we could read the platform logs file from edX in HDFS and detect abnormal users' behavior.

Likewise, after having achieved to detect fraudulent users, we analyzed our classified data stored in a dataset and studied the possibility of being able to predict this type of user through their interactions with the platform, in order to be able to detect them before that the course has finished. For this purpose, we have made use of Machine Learning using Python code to train and predict our dataset with different classification models, and we compared the results obtained with each of them calculating different evaluation metrics, finally obtaining what would be the optimal model in our specific case.

Palabras clave

Educación online, fraude, usuarios fraudulentos, Big Data, Spark, Hadoop, Aprendizaje Automático, SVM, Naive Bayes, Regresión Logística, Random Forest, dataset desbalanceado y técnicas de evaluación.

Keywords

Online education, cheating, fraudulent users, Big Data, Spark, Hadoop, Machine Learning, SVM, Naive Bayes, Logistic Regression, Random Forest, imbalanced dataset and evaluation metrics.

Agradecimientos

Antes de comenzar con la explicación de todo el trabajo realizado a lo largo de este curso, me gustaría agradecer todo el apoyo y la ayuda que me han ofrecido en todo momento tanto Estrella Pulido como Gonzalo Martínez. Gracias a ellos, he podido sacar adelante este trabajo, además de adquirir muchos más conocimientos, tanto en el área de Big Data como en el área de Machine Learning. Por último, agradecer enormemente a mi familia su apoyo incondicional, especialmente a mi abuelo, quien sin duda estaría muy orgulloso.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 Introducción

1.1 Motivación

Actualmente la educación online está a la orden del día a nivel mundial. Debido al increíble avance tecnológico de los últimos años, la mayor parte de los sectores se han visto afectados, incluyendo la educación. Para muchas personas el poder acceder de manera fácil y sin un coste muy elevado a una formación homologada y con prestigio supone una opción muy interesante para su expediente académico.

Los MOOC son un claro de ejemplo de cursos online que están cobrando una gran relevancia en este tipo de educación a través de internet. Estos cursos son impartidos por profesores que trabajan en instituciones docentes de cualquier parte del mundo como, por ejemplo, los profesores de la Universidad Autónoma de Madrid, que se encuentran actualmente impartiendo cursos en la plataforma edX.

Para conseguir atraer a un mayor número de personas a este tipo de formación, es importante mantener el prestigio de la calidad de sus certificados de cara al mundo laboral. El principal problema para este tipo de cursos no presenciales, es tener que tomar nuevas medidas para erradicar el posible fraude de alumnos a la hora de realizarlos, ya que las medidas tradicionales no sirven en estos casos.

Encontrándonos en este contexto, nuestro objetivo en este Trabajo Final de Grado, será estudiar posibles criterios para poder detectar usuarios fraudulentos a través de sus distintas interacciones con el curso online y analizar los resultados para determinar si es posible o no predecir si un usuario ha realizado fraude en base a sus interacciones.

Durante todo este proyecto, analizaremos el curso no presencial de la Universidad Autónoma de Madrid en la plataforma edX utilizando su almacenamiento de logs para cada usuario. A partir de estos datos, contaremos con una volumetría bastante pesada, por lo que desarrollaremos nuestro software haciendo uso de herramientas propias de Big Data. Finalmente, una vez tengamos nuestros usuarios clasificados dentro de nuestro dataset, comenzaremos con el análisis de los datos obtenidos utilizando técnicas propias de Machine Learning.

1.2 Objetivos

Durante este sub-apartado, destacaremos los objetivos principales de nuestro TFG. Éstos serán los siguientes:

- Estudiar nuevos criterios para la detección de fraude en cursos online que puedan reemplazar a los criterios tradicionales utilizados en formaciones presenciales.

- Automatización de la detección de fraude a través de los logs almacenados por la plataforma edX para conseguir clasificar un dataset con todos los alumnos del curso, según los criterios propuestos.

- Predicción de fraude en los alumnos utilizando el dataset clasificado y analizar los resultados obtenidos para poder determinar si es posible una predicción futura de usuarios fraudulentos partiendo de toda la información conseguida a lo largo del trabajo.

1.3 Organización de la memoria

Nuestra memoria se divide en los siguientes capítulos:

- **Estado del arte**. En este capítulo presentamos un estudio exhaustivo sobre el contexto actual de la educación no presencial y los problemas que esta nueva manera de formar a estudiantes conlleva para gestionar los posibles fraudes que pueden realizar algunos de ellos para aprobar el curso y obtener el título.

- **Tecnologías Big Data y Machine Learning**. En este capítulo definiremos algunos conceptos esenciales para entender el desarrollo del trabajo realizado y hablaremos muy brevemente sobre las herramientas utilizadas.

- **Desarrollo de detección de fraude**. A lo largo de todo este capítulo explicaremos los criterios establecidos para detectar el posible fraude realizado por los alumnos a partir de la información proporcionada por la plataforma edX, donde están contenidas todas las interacciones de cada uno de los usuarios con el curso.

- **Predicción y análisis de los resultados.** En este capítulo analizaremos los resultados obtenidos tras realizar predicciones de fraude en el dataset clasificado generado en nuestro trabajo. Estudiaremos los resultados de las medidas de evaluación de los modelos y haremos diferentes pruebas con los atributos del dataset para comparar resultados.

- **Conclusión.** Finalmente, en este capítulo presentaremos un análisis final de todo el trabajo realizado a lo largo del TFG y comentaremos las conclusiones a las que hemos llegado y el trabajo futuro que se podría realizar para perfeccionarlo.

2 Estado del arte

La evaluación es la base principal del sistema educativo actual [31], siendo el examen el método más utilizado a día de hoy. Según Foucault, un examen se podía definir de la siguiente manera:

"Combina las técnicas de la jerarquía que vigila y las de sanción que normaliza. Es una mirada normalizadora, una vigilancia que permite calificar, clasificar y castigar. Establece sobre los individuos una visibilidad a través de la cual se los diferencia y se los sanciona"

Foucault M., 1993 [16]

Tras la definición realizada por Foucault, podemos llegar a comprender que los estudiantes puedan llegar a experimentar un miedo a los exámenes completamente racional, ya que el resultado de éstos determinará en gran parte su futuro. Por ejemplo, un sentimiento negativo que se suele presentar en la mayor parte del alumnado a medida que se acercan las fechas estimadas de los exámenes, es el estrés [23], el cual constituye un grave problema tanto por el número de estudiantes a los que afecta como por el impacto negativo que conlleva en su rendimiento académico [22]. Este tipo de emociones solo consiguen que cualquier esfuerzo y dedicación del alumnado sea completamente inservible, provocando que la sensación que experimentan de fracaso e inseguridad aumente a tal nivel que decidan o bien evitar estudiar, o incluso no llegar a presentarse al examen aun quizás habiendo estudiado para ello.

Numerosos estudios psicológicos resaltan que muchos de los estudiantes afectados por este tipo de sentimientos, paradójicamente, no suelen ser malos estudiantes sino, por el contrario, demasiado perfeccionistas [40]. Este perfeccionismo y exigencia por obtener buenos resultados en las pruebas, junto con la presión y el estrés común en fechas destacables de exámenes, pueden ser detonantes para que ciertos alumnos que en un contexto normal no se plantearían quebrantar las normas, si se presenta la oportunidad, lleguen a cometer fraude académico. Este tipo de fraude también es conocido como "Triángulo del Fraude".

Rogers y Neumann propusieron en el año 2003 un modelo adaptativo que es utilizado como la base principal de la detección y gestión del fraude actual [43]. Este modelo, supone asumir que el motivo habitual por el cual una persona llega a cometer cierto tipo de fraude, ya sea económico o de algún otro tipo, es debido a encontrarse en una situación complicada donde el engaño cometido sería una respuesta adaptativa para salir de ese contexto. Hay que tener en cuenta que si el escenario de éste hubiera sido más favorable, no se vería sumido en la necesidad de llevar a cabo el engaño para obtener un beneficio a costa del perjuicio ajeno.

En 1961, el famoso criminólogo Donald R. Cressey propuso la teoría de que un evento fraudulento tiene lugar cuando se cumplen tres elementos principales; encontrarse en una situación de presión o con un motivo, tener la posibilidad de llevarlo a cabo y contar con una justificación viable para convencerse a uno mismo de las razones por las que comete dicho engaño [46]. Estos tres elementos claves son conocidos como "El Triángulo del Fraude", concepto que mencionamos anteriormente en el apartado.

Aunque en la mayor parte de los estudios sobre fraude académico se centran en el ámbito universitario, cabe destacar que no sólo este tipo de acciones son comunes en las universidades, sino que también se dan con frecuencia en la enseñanza secundaria [17]. Asimismo, esta práctica fraudulenta se encuentra muy extendida tanto en países emergentes como en países pertenecientes al primer mundo. Al revisar distintas encuestas realizadas a lo largo de diferentes décadas del país líder en el estudio de estas malas praxis, Estados Unidos, podemos comprobar un aumento del fraude académico a lo largo de los años, coincidiendo con la progresiva implantación de Internet en el sistema educativo [2].

Por ejemplo, en el año 2012 tuvo lugar un caso de gran relevancia de mala praxis dentro del ámbito educativo norteamericano en la Universidad prestigiosa de Harvard, la cual tuvo que hacer frente al mayor escándalo de fraude de su historia. Se abrió una investigación a 125 estudiantes de la universidad, los cuales supuestamente habían cometido actos fraudulentos a la hora de realizar un "home-test", es decir, un examen que se realizaba a distancia y de manera individual. Mathew Platt, profesor de esa asignatura dentro de la universidad, detectó que 125 de los 279 estudiantes matriculados en su asignatura, habían contestado de manera muy parecida a la prueba escrita [37].

Con todo este contexto histórico, podemos afirmar que el fraude académico ha estado muy presente a lo largo de la historia. Por este motivo, hay que asumir que con la implantación de nuevos sistemas educativos, gracias a los avances tecnológicos de las últimas décadas, este tipo de prácticas fraudulentas serán igual o más comunes dentro de las nuevas enseñanzas no presenciales. Los cursos online han conseguido transformar el sistema de enseñanza tradicional que predominaba hasta hoy en día. No sólo han conseguido atraer a un mayor número de estudiantes de cualquier parte del mundo por su posible acceso no presencial y por su interacción a tiempo real, sino que además es capaz de acercar una educación superior a personas de cualquier clase social y económica, por sus bajos costes.

Entre el 2002 y el 2012, hubo un aumento de títulos de licenciatura online otorgados en EE.UU de 71,000 títulos (de 4,000 a 75,000), lo que constituía un 5% de todos los títulos de Estados Unidos entregados a lo largo del 2012 [28]. A partir del 2013, el Gobierno federal calculó que alrededor del 27% de los estudiantes universitarios estaban cursando al menos un curso online. En 2018, 20 millones de nuevos estudiantes se inscribieron en al menos un MOOC, frente a los 23 millones que hubo en 2017. A pesar de que el número de estudiantes descendiese en 2018, se estima que aumentó el número de usuarios que realizaban cursos online de pago, lo que está dando lugar a que proveedores de MOOC como Coursera estén alcanzando unos ingresos récord (140 millones de dólares en 2018) [10]. A finales de 2018, más de 900 universidades de todo el mundo ofertaban cursos MOOC.

A raíz de esta nueva forma de estudio no presencial, comenzaron a surgir dudas e inquietudes sobre la acreditación y rigor con el que estos cursos certificaban a sus estudiantes. Esto se debía a que todas las técnicas aplicadas para acabar con el fraude académico hasta entonces, se basaban principalmente en estudios presenciales, lo que dejaba una ventana abierta en la educación online para que los alumnos pudieran llevar a cabo distintos tipos de engaños y trampas con el objetivo de conseguir el aprobado. Según un estudio del Servicio de Pruebas Educativas de Estados Unidos, hasta un 73% de los encuestados coinciden en que la mayoría de las personas realizan algún tipo de fraude a lo largo de sus carreras académicas [41].

Jarrod Morgan, fundador de ProctorU (plataforma utilizada para la supervisión de los estudiantes en la educación online), comentó en un artículo lo siguiente:

"*El trabajo de nuestra industria es garantizar que los programas online tengan el mismo prestigio y valor que un título en el campus. Si un programa de estudios universitarios se ve comprometido (…), la institución podría sufrir un daño significativo a su reputación. Si la sociedad en general pierde la confianza en la validez de las credenciales de la educación, su valor para los estudiantes se destruye*."

Jarrod Morgan, Febrero 2018 [26].

Como consecuencia de este problema, numerosas empresas y universidades comenzaron a investigar diferentes técnicas novedosas para combatir este nuevo tipo de fraude dentro de la educación, como por ejemplo, empresas tecnológicas como Respondus, ProctorU y Examity [2]. Su principal funcionalidad es conseguir detectar si un alumno está haciendo trampas o no al realizar un examen online. En general, el uso de este tipo de sistemas evita el uso de otras pestañas, programas y diversas funciones (como imprimir, compartir pantalla, etc.) en el ordenador en el que se está realizando el examen. En algunos casos, incluso llegan a utilizar un software de vídeo para supervisar al alumno mientras el examen está realizándose. Esta supervisión puede ser proporcionada por una grabación o por un supervisor en tiempo real.

El software de prevención de plagio también está disponible. Programas como SafeAssign o TurnItIn proporcionan un informe de originalidad en el que se marca el contenido que es similar a otra fuente y se le proporciona al instructor el contenido para realizar una comparación [38]. Estas herramientas buscan en las bases de datos de ensayos y otros trabajos escritos, así como en internet para proporcionar al instructor la mayor cantidad de información posible. El uso de este tipo de herramientas evidencia que la integridad de la educación online es una prioridad para los instructores universitarios y la administración.

En 2015, la universidad del MIT y Harvard publicaron un artículo en el que identificaban un nuevo algoritmo para la detección de usuarios fraudulentos en los cursos MOOC. Dicho algoritmo fue apodado con el nombre de CAMEO (*Copying Answers using Multiple Existences Online*) [1]. Este algoritmo, utilizado hoy en día por la plataforma online, fue desarrollado para detectar aquellos usuarios que pertenecen a una misma persona y son utilizados para conseguir los resultados de los exámenes. Este tipo de usuarios, trabajan asiduamente de la misma manera.

Una persona crea N usuarios distintos dentro de la plataforma que oferta el curso. Uno de estos usuarios será denominado usuario master, y será el que termine sacándose el certificado, y los N-1 usuarios restantes que pertenecen a la misma persona los utilizará únicamente para realizar las actividades, ya sean correcta o incorrectamente, y obtener las respuestas correctas de éstas. Estos usuarios nunca llegarán a terminar el curso, ya que a la persona detrás de este conjunto de usuarios no le conviene tener unas calificaciones desastrosas, y se refieren a ellos con el nombre de "harvester users" o "usuarios cosechadora" [35]. El algoritmo de detección CAMEO se basa en analizar la distribución de las diferencias en el tiempo entre las acciones de pares de usuarios enlazados por IPs duplicadas. Para identificar a los usuarios de CAMEO, el algoritmo se basa en un criterio bayesiano de las distribuciones de diferencia del timestamp [25].

Recientemente, se realizó un estudio sobre el caso concreto del algoritmo CAMEO para realizar la detección de los usuarios detectados como fraudulentos en tiempo real y sin

necesidad de utilizar el criterio de IPs duplicadas para determinar a los usuarios como CAMEO o no-CAMEO [35]. Este estudio se llevó a cabo a través del uso de técnicas de Machine Learning con un dataset formado por atributos pertenecientes a características del usuario, del problema realizado y de la interacción del usuario con dicho problema. El modelo aplicado fue un Random Forest, por su eficiencia a la hora de trabajar con diversos tipos de datos y con un alto número de atributos, a la vez que su rapidez a la hora de entrenar y predecir con un volumen pesado de datos. Durante la investigación, se determinó, con un acierto 96.64 % en la predicción de los usuarios CAMEO, la posibilidad de poder detectar estos usuarios sin necesidad de utilizar las IPs y en tiempo real [25].

Asimismo, en 2019, tres estudiantes de la universidad del MIT publicaron un estudio relacionado sobre el algoritmo que implementaron en el año 2015 para detectar usuarios fraudulentos en los cursos MOOC, que hemos mencionado anteriormente [3]. En este último artículo se demostraba, no sólo que los modelos utilizados para detectar usuarios fraudulentos que usaban el método de CAMEO se detectaban correctamente, sino que además ese modelo entrenado con el conjunto de datos también detectaba de manera satisfactoria otro tipo de fraude denominado "colaboración no autorizada". Este método consiste en que dos usuarios distintos compartan resultados entre ellos, colaborando para poder finalizar el curso sin tener que adquirir los conocimientos requeridos.

El objetivo principal de este estudio era poder desarrollas técnicas de detección de usuarios fraudulentos sin asumir que cumplen una serie de patrones concretos, es decir, intentar conseguir no depender de patrones específicos de comportamiento sino detectar comportamientos anómalos generales de los usuarios. Para conseguir este objetivo, plantearon la hipótesis de que la detección de fraude de un tipo de engaño en específico valdría para construir un modelo capaz de detectar otros tipos de engaño. Partiendo de un modelo de referencia para detectar a los usuarios que cumplen el método de "colaboración no autorizada", compararon los resultados del modelo entrenado para detectar CAMEO prediciendo a usuarios colaboradores y probar si efectivamente podría servir para detectar de manera genérica cualquier tipo de fraude. El resultado final obtenido demuestra que un usuario de tipo colaborador tiene 4,5 veces más probabilidades de ser clasificado como "positivo" que como "negativo".

Algunas nuevas investigaciones sugieren que el fraude puede abordarse mediante el uso de datos biométricos para identificar a los estudiantes según sus características fisiológicas y de comportamiento [7]. La biometría comúnmente, utiliza rasgos sofisticados como el género, la edad, la altura, el peso y la etnicidad, características fisiológicas como la cara, los ojos y las manos y características de comportamiento tales como pulsaciones de teclas, firma, movimiento del ratón, voz, marcha y pulso para reconocer a los individuos. Se pueden combinar dos o más de los datos biométricos enumerados para mejorar la precisión del reconocimiento [20]. Los sistemas actualmente utilizados en las universidades que hacen uso de una autenticación biométrica son Securexam Student (SES), Webassessor y ProctorU. Por ejemplo, la plataforma Webassessor de Kryterion utiliza imágenes de rostros capturadas por cámaras web y rasgos biométricos tales como las pulsaciones de las teclas (estilos de escritura) capturados por el software, para autenticar al examinado, y en el caso de que se detecte alguna anomalía, alertar a los supervisores [20].

3 Tecnologías Big Data y Machine Learning

Durante el desarrollo de este proyecto hemos hecho uso de dos nuevas tecnologías: Big Data y Machine Learning. Para implementar todo el proceso de detección de usuarios fraudulentos a través del procesamiento de los logs generados por el curso edX, hemos utilizado un clúster Hadoop. El ecosistema de Hadoop trata de un complejo entorno de trabajo para facilitar la gestión de datos masivos. La base de este ecosistema consta principalmente de un sistema de ficheros distribuidos (HDFS) y un procesamiento de grandes cantidades de datos a través de MapReduce [5]. A partir de esta base, Apache ha desarrollado frameworks para cada necesidad funcional dentro del ecosistema, como podemos ver en la Figura 1.

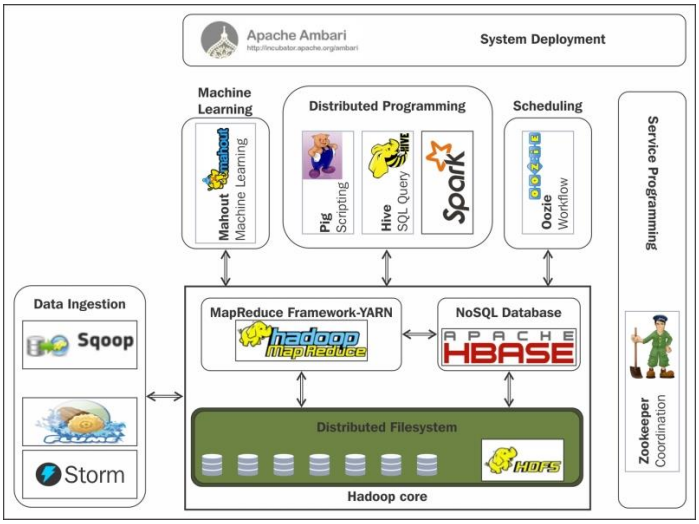


Figura 1 – Ecosistema Hadoop

Los frameworks utilizados en nuestro proyecto han sido: Apache Spark, HDFS y YARN, de los cuales haremos una breve explicación sobre sus arquitecturas y la funcionalidad de cada uno de ellos [14].

HDFS es un sistema de ficheros estructurados en bloques donde cada fichero se divide en bloques de un tamaño predeterminado (128 MB en Apache Hadoop 2.0 y 64MB en apache Hadoop 1.0) [21]. Estos bloques generados se almacenan dentro del clúster, el cual puede estar formado por una o varias máquinas. HDFS consta de una arquitectura Maestro/Esclavo, donde un clúster está formado por un solo NameNode (nodo maestro) y el resto de nodos serían los DataNodes (nodos esclavos), los cuales se distribuyen en varias máquinas. En la Figura 2, podemos ver un ejemplo de la arquitectura HDFS.

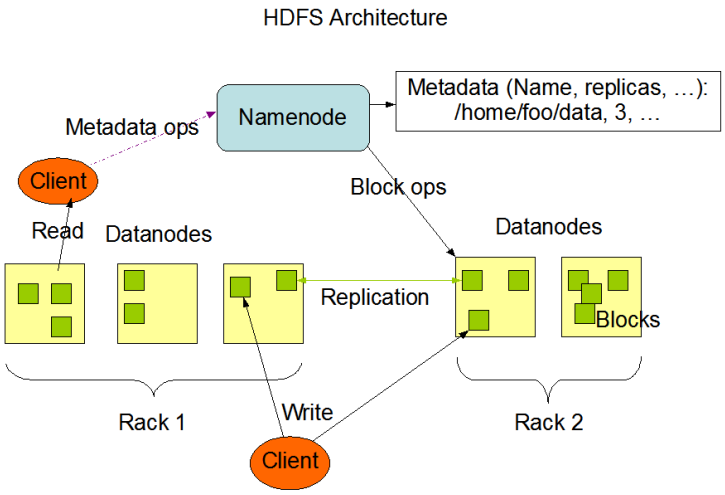


Figura 2 – Arquitectura de HDFS

El NameNode es el nodo maestro dentro de la arquitectura HDFS y se encarga de la administración y mantenimiento de los bloques presentes en los nodos esclavos del clúster. Los datos únicamente residen en los DataNodes, permitiendo que el NameNode sea un servidor con una alta disponibilidad. Este servidor se encarga también de la gestión del espacio de nombres del sistema de archivos y del control de permisos a los ficheros por parte de los clientes. Los DataNodes, por el contrario, son servidores de bloques que almacenan los datos en archivos locales. Éstos son responsables de servir las peticiones de lectura y escritura de los clientes del sistema de archivos, además de realizar la creación, eliminación y replicación de bloques a partir de la instrucción del NameNode.

Como hemos comentado anteriormente, la base principal del ecosistema Hadoop consistía en el uso de HDFS y MapReduce. Sin embargo, en la versión Apache Hadoop 2.0, MapReduce fue sustituido por Apache YARN que es el que utilizaremos en nuestro ecosistema. Apache YARN es el nuevo framework que se encarga de administrar los recursos de memoria y CPU a los diferentes procesos. Asimismo, maneja y programa la solicitud de recursos de la aplicación y ayuda al proceso a ejecutar la solicitud [4]. Finalmente, MapReduce 2.0 se quedó como una aplicación distribuida que se ejecuta sobre YARN para el procesamiento de datos. En la Figura 3, podemos ver una comparativa de la arquitectura de Hadoop entre la versión 1.0 y 2.0.

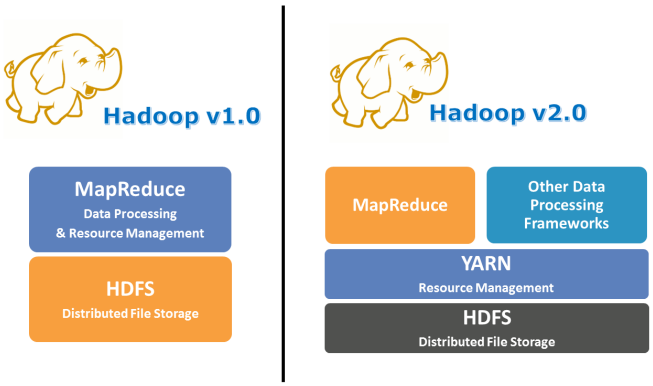


Figura 3 – Comparativa de arquitecturas Hadoop 1.0 y Hadoop 2.0

A pesar de ser MapReduce un framework de procesamiento de datos, en nuestro proyecto hemos llevado a cabo el desarrollo mediante el framework Apache Spark, el cual procesa los datos a una velocidad superior. Apache Spark es un framework de código abierto utilizado tanto para el procesamiento de datos masivos, como para datos generados en tiempo real [6]. La principal característica de Apache Spark es que consta de un clúster con una arquitectura In-Memory (IM), consiguiendo ejecutarse hasta 100 veces más rápido que MapReduce para el procesamiento de datos a gran escala. Este motor de procesamiento distribuido es responsable de la gestión y monitorización de aplicaciones formadas por múltiples tareas de procesamiento sobre varias máquinas que conforman un clúster.

Al igual que HDFS, Spark consta de una arquitectura Maestro/Esclavo. En la terminología del framework, se denominará "Driver" al nodo maestro y "Executors" a los nodos esclavos. El Driver es el nodo encargado de la gestión, distribución y mantenimiento de toda la información necesaria durante la ejecución del proceso. Los Executors, se encargarán únicamente de ejecutar las tareas asignadas por el driver y devolver a éste el resultado final. En la Figura 4, podemos visualizar la arquitectura de Apache Spark.

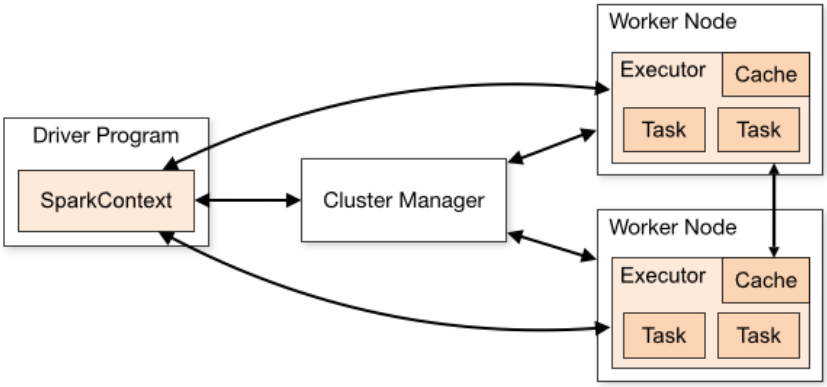


Figura 4 – Arquitectura de Apache Spark

Vemos que en todo momento se mantiene conexión durante la ejecución con el Cluster Manager, el cual se encarga de asignar los recursos del sistema y que en nuestro caso será Apache YARN, como hemos comentado anteriormente. Cuando procedamos a la carga de un conjunto de datos para ser procesado por Apache Spark, habrá tres estructuras posibles que podremos utilizar, que son: RDDs, DataFrames, o DataSets.

Un RDD es sinónimo de conjuntos de datos distribuidos resilientes [39]. Es una colección de particiones de sólo lectura de registros. RDD es la estructura de datos fundamental de Spark, permitiendo a un programador realizar cálculos en memoria de grupos grandes de datos, de manera tolerante a fallos.

En un DataFrame a diferencia de un RDD, los datos se organizan en columnas con nombre como si se tratase de una tabla en una base de datos relacional. Los DataFrames en Spark permiten a los desarrolladores trabajar con una estructura impuesta a una colección distribuida de datos, lo que permite una abstracción a más alto nivel [11]. Los DataSets en Apache Spark son una extensión de la API DataFrame, que proporciona una interfaz de programación orientada a objetos, introducida en Spark a partir de la versión 1.6 [11].

La utilización de cada una de las diferentes estructuras está muy ligada a las necesidades específicas de cada proyecto. En nuestro caso, optamos por la utilización de DataFrames, debido a que ofrecen una vista y una estructura personalizada de los datos, ahorran espacio y se ejecutan a alta velocidad [11].

Tras conseguir llevar a cabo la detección de usuarios fraudulentos dentro del curso, decidimos realizar, dentro de nuestro proyecto, una segunda fase enfocada al análisis y predicción de resultados para estudiar la posibilidad de predecir este tipo de usuarios en tiempo real y utilizando un dataset con variables diferentes a las que aparecen dentro de los ficheros de logs de la plataforma edX. Para conseguir desarrollar esta segunda etapa del proyecto, nos enfocamos en el campo de Machine Learning.

En esta segunda fase, hemos realizado nuestra implementación utilizando el lenguaje de programación Python para el uso de la librería Sklearn**.** La librería Sklearn contiene herramientas simples y eficientes para analizar y tratar con datos, como en el caso de las áreas de Data Mining y Data Analysis [45]. Gracias a esta librería importaremos los algoritmos de clasificación Naive Bayes, Regresión Logística, SVM y Random Forest.

Asimismo, tras obtener los resultados de clasificación de cada uno de estos modelos, importaremos también de la API Sklearn funciones que nos servirán de ayuda para el cálculo de métricas de evaluación como Precision, Recall, F1-Score, matriz de confusión, la curva ROC y el AUC score. Gracias al cálculo de estas métricas, podremos analizar más exhaustivamente cada resultado de clasificación. A continuación, definiremos brevemente cada una de las métricas utilizadas comenzando por la matriz de confusión.

La matriz de confusión consta en nuestro proyecto de una matriz 2x2, ya que hay únicamente 2 clases, en la cual cada fila representa a las instancias en la clase real y en las columnas las instancias predichas [34]. En la Figura 5, podemos ver representada la matriz.

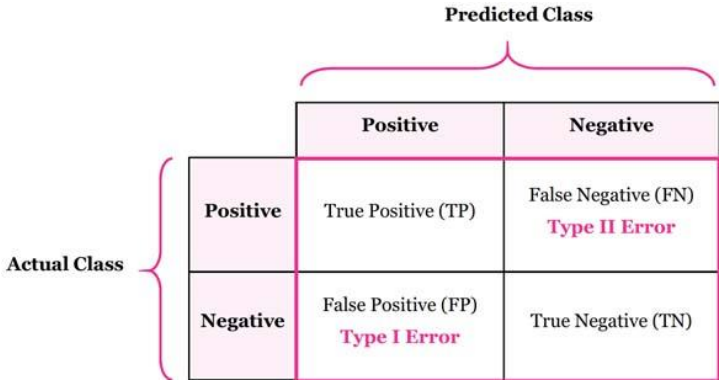


Figura 5 – Matriz de confusión

Como podemos observar, en cada casilla se marca una categoría distinta: TP, FP, FN y TN. TP hace referencia a todos los usuarios positivos que se predijeron correctamente, por el otro lado, FN hace referencia a todos los usuarios positivos que el modelo clasificó como negativos. En el caso de FP, representa a todos los usuarios negativos que el modelo clasificó como positivos, y por último, TN consta de todos los usuarios negativos que el modelo clasificó correctamente.

Gracias al cálculo de esta matriz podemos aplicar las métricas de evaluación Precision, Recall y F1-Score. Las siguientes fórmulas representan el cálculo de las métricas Precision y Recall:

$$Recall = \frac{TP}{TP + FN} \qquad Precision = \frac{TP}{TP + FP}$$

Podemos apreciar que ambas fórmulas son muy parecidas pero no calculan lo mismo. Recall divide el valor de usuarios positivos clasificados correctamente entre la suma de éstos con los usuarios positivos clasificados por el modelo como negativos. Por lo tanto, esta métrica realiza el cálculo del porcentaje de cuántos usuarios han sido clasificados como positivos por el modelo partiendo del total de usuarios positivos reales del dataset. Por otro lado, Precision divide también el valor de los usuarios positivos clasificados correctamente, pero esta vez el denominador es distinto ya que se divide entre la suma de éstos con los usuarios negativos que fueron clasificados como positivos por el modelo. Por lo que esta métrica calcula el porcentaje de usuarios clasificados como positivos por el modelo que se han clasificado correctamente [34]. Finalmente, la métrica F1-Score se calcularía con la siguiente fórmula:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Como podemos ver, combina el valor de ambas métricas explicadas anteriormente, convirtiéndose en una métrica considerada como una media armónica empleada para la determinación de un valor único ponderado entre Precision y Recall [34].

La curva ROC es otra de las herramientas más utilizadas dentro del análisis de los resultados de modelos de clasificación en Machine Learning. Esta curva es una representación gráfica de la tasa de verdaderos positivos (TPR) frente a la tasa de falsos positivos (FPR) en diferentes umbrales de clasificación, mostrándonos el rendimiento de un modelo de clasificación en todos los umbrales [33]. En la Figura 6, podemos ver representada una curva ROC, donde VP equivale a TP (Verdadero Positivo ≡ True Positive).

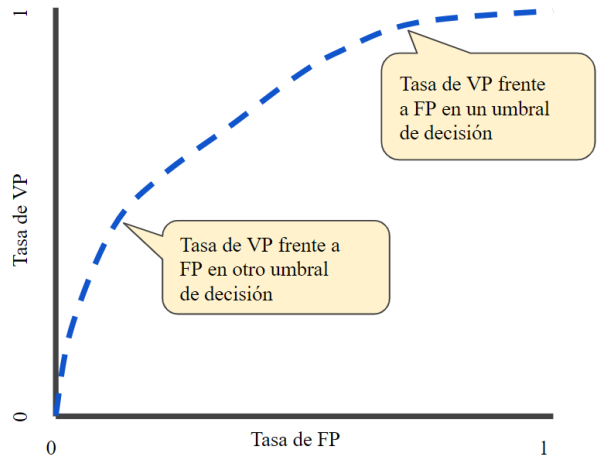


Figura 6 – Curva ROC

A partir de este gráfico podremos resolver el AUC Score, ya que éste hace referencia al área bajo la curva ROC. Esto significa que el AUC mide toda el área bidimensional por debajo de la curva [33]. Esta métrica nos indica cuánto es capaz el modelo de distinguir entre clases, es decir, predecir los positivos como positivos y los negativos como tal.

Uno de los principales problemas que tuvimos a la hora de realizar esta segunda fase era que contábamos con un dataset totalmente desbalanceado debido al caso en concreto de nuestro trabajo. Trabajar con este tipo de datasets es bastante complicado ya que cualquier modelo de clasificación realiza los entrenamientos con conjuntos de datos donde hay claramente una clase mayoritaria y otra minoritaria, inclinándose hacia la primera. Para evitar este problema, utilizamos la API Imblearn de Python. Esta API contiene numerosas librerías que nos facilitaron funciones para trabajar de manera más sencilla con este tipo de datasets tan complejos [24]. En nuestro caso en concreto, hemos dado uso a métodos como Undersampling, Oversampling y el modelo de clasificación Random Forest que balancea automáticamente a la hora de entrenar con un conjunto de datos desbalanceado.

La técnica Oversampling que utilizaremos será Random Oversampling. Esta técnica implica aumentar el número de registros de la clase minoritaria duplicando al azar algunos ejemplos de éstos y agregarlos al conjunto de datos de entrenamiento [47]. Los ejemplos del conjunto de datos de entrenamiento se seleccionan aleatoriamente con reemplazo, es decir, que se va a duplicar un registro del conjunto de datos existente para aumentar el número de ejemplos de esa clase. Ese registro original que fue duplicado, no se eliminará por lo que se podrá volver a duplicar en futuras ocasiones. Debido a esto, puede que a la hora de realizar el entrenamiento cuente con un conjunto de datos balanceado pero que nuestro modelo sufra un sobreajuste, y a la hora de predecir el conjunto de datos de prueba no obtenga un buen resultado.

Por último, la técnica de Undersampling utilizada es Random Undersampling. Esta técnica realizaría una selección aleatoria de registros dentro de la clase mayoritaria e irá eliminando la información seleccionada hasta generar un conjunto de datos de entrenamiento totalmente balanceado entre las clases, provocando una pérdida de información sobre la clase mayoritaria. En la Figura 7, podemos visualizar el comportamiento de una y otra técnica, mencionada anteriormente.

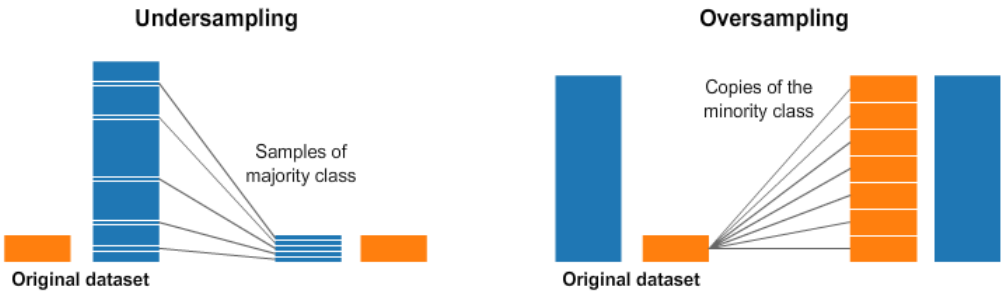


Figura 7 – Comparativa de las técnicas UnderSampling y OverSampling

4 Desarrollo de detección de fraude

En este apartado trataremos en profundidad el estudio de las interacciones de cada usuario con un curso edX, para conseguir detectar a usuarios sospechosos de realizar un posible fraude. Disponemos de dos ficheros en los que están almacenados todos los logs de los usuarios registrados en el curso "Jugando con Android" de la Universidad Autónoma de Madrid a lo largo del año 2015. Estos dos ficheros guardan todos los datos relevantes de cada uno de los eventos que realizan los usuarios, que necesitaremos más adelante.

Decidir clasificar a un usuario como fraudulento o no, depende de una serie de criterios que decidimos implementar tras la búsqueda realizada en la etapa de investigación. Los criterios que determinarán esta clasificación son los siguientes:

Criterio 1: Se hará referencia a un par de usuarios como posibles usuarios fraudulentos, tras detectar que éstos comparten la misma IP al realizar el mismo ejercicio.
Cogemos el par de usuarios detectado, y comprobamos si se cumple alguno de los siguientes resultados:

a. Si el usuario que realizó el ejercicio primero tiene como resultado del ejercicio "Correcto", y el usuario que lo realizó más tarde tiene como resultado "Correcto".
b. Si el usuario que realizó el ejercicio primero tiene como resultado del ejercicio "Incorrecto", y el usuario que lo realizó más tarde tiene como resultado "Correcto".

Si se da alguno de los casos mencionados anteriormente, podemos considerar esa pareja de usuarios como posible sospechosa de haber realizado un fraude.

Tanto el primero como el segundo resultado tratan de encontrar pares de usuarios en el que un usuario sea de tipo "*cosechador*", es decir, pruebe a responder a las preguntas y termine obteniendo el resultado correcto (da igual que acierte o que falle), y el usuario "*maestro*" que sería aquel usuario que es el último en realizar la prueba de la pareja y obtiene el resultado como correcto gracias al usuario cosechador mencionado anteriormente.

Este criterio no sería un criterio muy sólido, ya que podría darse el caso de que dos usuarios compartieran la misma red, por ejemplo, que realicen el ejercicio dentro de la universidad o en una residencia de estudiantes, y por lo tanto podrían tener la misma IP pública. De igual manera, un estudiante podría esconderse muy fácilmente de este criterio a través de una conexión proxy para modificar su IP. Por este motivo, no nos vamos a basar únicamente en este criterio para decidir si un usuario es sospechoso de fraude o no, sino que deberá de cumplir alguno (o ambos) de los siguientes criterios determinados.

Criterio 2: En este criterio detectaremos los usuarios sospechosos de fraude que han realizado los exámenes del curso, sin haber accedido en ningún momento al contenido adjuntado, ya sean vídeos o documentos, contestando de manera correcta a todos los ejercicios.

Criterio 3: En este criterio, tomaremos como dato clave el timestamp de realización de los ejercicios por parte de los usuarios. Detectaremos parejas de usuarios sospechosas de fraude, como en el criterio 1, comparando los timestamp de cada uno de los ejercicios.

La diferencia entre estos timestamps deberá de ser menor o igual a 30 segundos, para poder asegurar que realizaron el mismo ejercicio con una diferencia de tiempo muy pequeña. De manera similar al primer criterio, comprobaremos si se dan los siguientes resultados:

a. Si el usuario que realizó el ejercicio primero tiene como resultado del ejercicio "*Correcto*", y el usuario que lo realizó más tarde tiene como resultado "*Correcto*".
b. Si el usuario que realizó el ejercicio primero tiene como resultado del ejercicio "*Incorrecto*", y el usuario que lo realizó más tarde tiene como resultado "*Correcto*".

Si se da alguno de los casos mencionados anteriormente, podemos considerar esa pareja de usuarios como posible sospechosa de haber realizado un fraude por la misma razón que en el criterio 1.

Tras haber consolidado todos los criterios que utilizaremos en la detección de usuarios que han llevado a cabo algún tipo de engaño, empezaremos a explicar detalladamente todo el proceso realizado para poder desarrollar la comprobación de los tres criterios anteriores con los datos que se encuentran en los ficheros logs.

Para poder clasificar a cada uno de los usuarios disponemos de un dataset, previamente implementado, con numerosas características de cada uno de estos usuarios respecto a sus interacciones con el curso. Gracias a esto, podremos más adelante analizar si es posible o no la predicción de este tipo de estudiantes en los cursos no presenciales.

En los siguientes sub-apartados podremos ver partes del código que hemos desarrollado para cumplir con la funcionalidad requerida en el proyecto, pero en el Anexo A, se puede ver de manera más exhaustiva cómo se ha llevado a cabo toda la parte del desarrollo y la ejecución del código.

4.1 Limpieza y extracción de datos

Los ficheros que almacenan los logs del curso, se encuentran implementados en formato JSON como podemos visualizar en la Figura 8.

Figura 8 – Fichero de logs en formato JSON

Contamos con dos ficheros de 87MB y 200MB respectivamente, aunque ciertos datos están duplicados. El primer fichero, que cuenta con una volumetría de 87MB, contiene todos los eventos realizados por cada usuario. Cada usuario tiene un array de Eventos, que dependiendo del tipo de evento que sea (determinado por el campo *evento*), se informa a los campos relevantes para éste en específico, es decir, en el caso de que un evento sea la interacción del usuario con un documento, se informa del título del documento y el timestamp del momento en que accedió a éste. El segundo fichero de 200MB contiene la misma información de cada usuario, exceptuando que este fichero también almacena todas las interacciones con los videos del curso, eventos que no tenía almacenados el primer fichero.

El resto de eventos se duplicarían entre ambos, por lo que podríamos determinar trabajar únicamente con el fichero de 200MB ya que contiene todos los eventos y así no tendríamos que tener en cuenta la duplicación de datos. Sin embargo, tenemos que utilizar el primer fichero debido a que éste contiene almacenados todos los eventos de problemas que han realizado cada uno de los usuarios con el campo *IP* informado, el cual no se rellena en ningún momento en el segundo fichero.

Por lo tanto, finalmente trabajaremos con estos dos ficheros teniendo presente que la mayor parte de los eventos se encuentran duplicados, exceptuando los relacionados con vídeos y problemas (se duplica toda la información de estos eventos, salvo el campo *IP*). Partiendo de este análisis de los datos que tendremos como entrada en nuestro proceso, realizaremos el siguiente tratamiento concreto de éstos.

Comenzaremos cargando cada uno de estos ficheros en un dataframe distinto y realizaremos una limpieza de los datos; eliminando aquellos usuarios cuyo identificador tome un valor nulo o venga vacío, además de aquellos eventos almacenados como *error_json*, ya que esto significa que por algún motivo ese evento ha dado un problema al almacenar su log.

Tras haber realizado la limpieza de los datos, utilizaremos los dos dataframes cargados para generar la siguiente estructura de datos:

- Un vista temporal que almacene todos los eventos, con un campo *id_usuario* para identificar al usuario que lo realizó.
- Un vista temporal que almacene todos los resultados de cada ejercicio realizado en un problema, con un campo *id_evento* para identificar a qué evento pertenecen esos resultados.

Para poder llevar a cabo esta extracción, limpieza y creación de la estructura de datos hemos definido una función en Scala denominada *loadFromJson*. Ésta abarca toda la funcionalidad mencionada anteriormente.

A partir de nuestra variable de tipo SparkSession, cargaremos los ficheros en formato JSON a dos dataframes diferentes, como vemos en la Figura 9.

```
var resource_data = spark.read.json(s"$jsonVid")
var resource_data_ip = spark.read.json (s"$pathJsonIP")
```

Figura 9 – Lectura de ficheros JSON en Spark

Como podemos intuir, gracias a los nombres de los dataframes, *resource_data* es el dataframe que contiene el fichero de logs incluyendo los eventos relacionados con vídeos y *resource_data_ip* contiene el fichero de logs que almacena la ip del usuario a la hora de realizar un problema. Tras cargar los datos, eliminaremos de ambos dataframes todos aquellos registros en los que el identificador de usuario tenga un valor nulo o vacío, como podemos ver en la Figura 10.

```
resource_data = resource_data.filter(resource_data("Usuario") =!= "" &&
  resource_data("Usuario").isNotNull)

resource_data_ip = resource_data_ip.filter(resource_data_ip("Usuario") =!= "" &&
  resource_data_ip("Usuario").isNotNull)
```

Figura 10 – Eliminación de datos no válidos del dataframe

A partir del dataframe sobreescrito *resource_data*, seleccionamos todos los eventos exceptuando los relacionados con problemas y aquellos que fueron almacenados como *error_json*, guardándolos en un nuevo dataframe llamado *dfEventos*, el cual almacenará todos los eventos que utilizaremos a lo largo de nuestro proceso. Seleccionaremos dentro del campo *Eventos* de nuestro dataframe, que es de tipo array, cada uno de los eventos como un nuevo registro y renombraremos el nombre del campo *Usuario* por *id_usuario*. Asimismo, como en este nuevo dataframe vamos a unir los datos de *resource_data* y *resource_data_ip*, tendremos que homogeneizar ambos esquemas, ya que son distintos.

Tras haber analizado cada uno de los esquemas, podemos observar que *resource_data* cuenta con seis campos que no se encuentran en el dataframe *resource_data_ip*, ya que son campos que se utilizan solamente en los eventos vinculados con vídeos, mientras que, *resource_data_ip* contiene el campo *IP* y el dataframe *resource_data* no.

Para tener un único esquema en el nuevo dataframe que pueda almacenar todos los datos de ambos, añadiremos estos nuevos campos que faltan a cada dataframe como columnas con un valor por defecto. En la Figura 11, podemos ver la implementación de lo comentado anteriormente.

```
dfEventos = dfEventos.withColumnRenamed("Usuario","id_usuario")
  .filter(dfEventos("evento") =!= "problem_check" && dfEventos("evento") =!= "error_json")
  .withColumn("ip",lit("NOT_IP"))

var df_probsIP = resource_data_ip.select(resource_data_ip("Usuario"),
    explode(resource_data_ip("Eventos")).as("eventos"))
  .select("Usuario","eventos.*")

df_probsIP = df_probsIP.withColumnRenamed("Usuario","id_usuario")
  .filter(df_probsIP("evento") === "problem_check")
  .withColumn("currentTime",lit("NOT_VIDEO")).withColumn("id_video",lit("NOT_VIDEO"))
  .withColumn("new_speed",lit("NOT_VIDEO")).withColumn("new_time",lit("NOT_VIDEO"))
  .withColumn("old_speed",lit("NOT_VIDEO")).withColumn("old_time",lit("NOT_VIDEO"))
```

Figura 11 – Esquema único para dataframe de eventos

Hemos creado un dataframe auxiliar *df_probsIP*, para almacenar los datos provenientes del dataframe *resource_data_ip*, añadiendo los seis campos que contiene *resource_data* y recogiendo de este dataframe únicamente los eventos que contengan las interacciones con problemas, ya que son los que nos interesan y así a la hora de unir ambos dataframes no obtendremos duplicados.

Una vez tengamos los eventos con el esquema deseado y sin duplicados, uniremos ambos dataframes, guardando el resultado de la unión en el dataframe *dfEventos*, y añadiremos una nueva columna con nombre *id_evento*, que contendrá un identificador único para cada evento almacenado, ya que en los datos extraídos de los ficheros no existe. Estos dos pasos, podemos verlos implementados en la Figura 12.

```
dfEventos = dfEventos.union(df_probsIP)
  .withColumn("id_evento", monotonically_increasing_id()+1)
```

Figura 12 – Unión de dataframes e inserción de un nuevo campo identificador

Este dataframe que contiene todos los eventos, cuenta con un campo denominado *resultados* que es de tipo array y contiene cada uno de los resultados de los ejercicios realizados en el problema del evento. Para mantener estos datos, los almacenaremos en un dataframe a parte y eliminaremos este campo de tipo array del dataframe *dfEventos*.

El dataframe que contiene todos los resultados se llama *dfResultados*, y mantiene una relación de 1-N con el dataframe *dfEventos* ya que cada evento puede tener N resultados, sin embargo, un resultado solo puede pertenecer a 1 evento. Por este motivo, mantendremos el campo *id_evento* en el dataframe para identificar a qué evento pertenece ese resultado y eliminaremos el campo de *resultados* en el dataframe *dfEventos*.

Por último, añadiremos un nuevo campo *id_resultado* para tener un identificador único por registro, ya que al igual que los registros de los eventos, no se extrajo ningún identificador de los ficheros cargados. Cada uno de estos pasos, podemos verlos realizados en la Figura 13.

```
val dfResultados = dfEventos.select(dfEventos("id_evento"),
    explode(dfEventos("resultados")).as("resultados"))
  .select("id_evento","resultados.*")
  .withColumn("id_resultado", monotonically_increasing_id()+1)
```

Figura 13 – Creación del dataframe de resultados

En la Figura 14, podemos observar cómo se crean las dos vistas temporales a partir de los dataframes que hemos generado, para poder utilizar estos datos más adelante como si se tratasen de dos tablas estructuradas.

```
dfEventos.drop("resultados").createOrReplaceTempView(s"$events_view")
dfResultados.createOrReplaceTempView(s"$results_view")
```

Figura 14 – Creación de las vistas temporales de eventos y resultados

4.2 Implementación de los criterios

Para implementar los tres criterios que hemos explicado al comienzo de este apartado, haremos uso de SparkSQL, para utilizar consultas SQL directas hacia las vistas temporales creadas en el anterior sub-apartado.

A lo largo de este sub-apartado realizaremos una explicación detallada de la implementación llevada a cabo para cubrir la funcionalidad necesaria y detectar aquellos usuarios fraudulentos que cumplen los criterios que hemos determinado.

4.2.1 **Usuarios sospechosos por IP**

Este primer criterio trata de identificar aquellos usuarios distintos que comparten la misma IP y que hayan realizado el mismo problema. Realizaremos una query SQL sobre únicamente los eventos de tipo problema, en la que compararemos cada uno de los id_usuario diferentes y filtraremos por aquellos que tengan el mismo valor de IP. Asimismo, calcularemos un nuevo campo que será igual a la diferencia entre los timestamp de cuando realizaron el problema cada uno de los usuarios para saber cuál de ellos fue el primero en responder al problema. Tras haber seleccionado este grupo de par de usuarios, filtraremos aquellos pares de usuarios que cumplan los siguientes casos:

- Si la diferencia entre los timestamp es un valor negativo, es que el usuario_1 ha realizado antes el problema que el usuario_2, por lo tanto filtramos para comparar el mismo ejercicio dentro del mismo problema y sólo seleccionamos a los pares de

usuarios que, o bien hayan respondido lo mismo al ejercicio, o el usuario_1 respondiese mal pero el usuario_2 tuviese correcto el ejercicio. Este último filtro recogería aquellos posibles usuarios "*cosechadora*" como comentamos en apartados anteriores.

- Si la diferencia entre los timestamp es un valor positivo, es que el usuario_2 ha realizado antes el problema que el usuario_1, por lo tanto filtramos para comparar el mismo ejercicio dentro del mismo problema y sólo seleccionamos a los pares de usuarios que, o bien hayan respondido lo mismo al ejercicio, o el usuario_2 respondiese mal pero el usuario_1 tuviese correcto el ejercicio.

La consulta SQL que devuelve los pares de usuario que cumplen este primer criterio comentado, se puede visualizar en la Figura 15.

```
val selectQuery: String =
    s"""
    WITH users_probs AS
        (select distinct id_usuario, $events_view.id_evento, CAST(tiempo as timestamp) as timestamp,
        id_problema, correcto, id_ejercicio, respuesta, ip
        FROM $events_view, $results_view
        WHERE upper(evento) = upper('problem_check')  and $events_view.id_evento = $results_view.id_evento) ,
    t_user1 AS
        (select id_usuario as id_usuario1, id_evento as id_evento1,  timestamp as timestamp1,
        id_problema as id_problema1,correcto as correcto1, id_ejercicio as id_ejercicio1,
        respuesta as respuesta1, ip as ip1
        FROM users_probs),
    t_user2 AS
        (select id_usuario as id_usuario2, id_evento as id_evento2, timestamp as timestamp2,
        id_problema as id_problema2, correcto as correcto2, id_ejercicio as id_ejercicio2,
        respuesta as respuesta2, ip as ip2
        FROM users_probs)
    SELECT id_usuario1, id_usuario2
    FROM
        (SELECT id_usuario1, id_evento1, timestamp1, id_problema1, correcto1, id_ejercicio1,
        respuesta1, ip1, id_usuario2, id_evento2, timestamp2, id_problema2, correcto2,
        id_ejercicio2, respuesta2, ip2, CAST(timestamp1 as long) - CAST(timestamp2 as long) as diffTime
        FROM t_user1
        INNER JOIN t_user2
        WHERE t_user1.id_usuario1 > t_user2.id_usuario2 and ip1 = ip2)  t_aux
    WHERE
    ( diffTime < 0 and id_problema1 = id_problema2 and id_ejercicio1 = id_ejercicio2  and
        (respuesta1 = respuesta2 OR (upper(correcto1) = upper('False') AND upper(correcto2) = upper('True') ) ) )
    OR ( diffTime >=0 and id_problema1 = id_problema2 and  id_ejercicio1 = id_ejercicio2  and
        (respuesta1 = respuesta2 OR (upper(correcto2) = upper('False') AND upper(correcto1) = upper('True') ) ) )
    """
```

Figura 15 – Consulta para seleccionar pares de usuarios que cumplen el criterio 1

Una vez tenemos implementada la query, ejecutaremos esta consulta utilizando SparkSQL y obtendremos como resultado un dataframe con pares de usuarios que cumplen el criterio 1. Utilizaremos este dataframe para agrupar por cada id_usuario y así saber cuántas veces cumple este criterio cada usuario en concreto. Para tener guardado este resultado, que utilizaremos más adelante, crearemos una vista temporal que denominaremos *table_suspectsIP*, como podemos ver en la Figura 16.

```
val df_suspect_IP = spark.sql(s"$selectQuery")

val count_users2 = df_suspect_IP.groupBy("id_usuario2")
                    .count()
                    .withColumnRenamed("count","count_users2")

val count_users1 = df_suspect_IP.groupBy("id_usuario1")
                    .count()
                    .withColumnRenamed("count","count_users1")

val users_num_iter_suspectIP = count_users2.alias("df_count_users2")
  .join(count_users1.alias("df_count_users1"),
    $"df_count_users1.id_usuario1" === $"df_count_users2.id_usuario2", "outer")
  .withColumn("count_users2",
    when(col("count_users2").isNull,lit(0)).otherwise(col("count_users2")))
  .withColumn("count_users1",
    when(col("count_users1").isNull,lit(0)).otherwise(col("count_users1")))
  .withColumn("sum_iters",
    List(col("count_users1"), col("count_users2")).reduce(_+_))

users_num_iter_suspectIP.createOrReplaceTempView("table_suspectsIP")
```

Figura 16 – Creación de la vista temporal table_suspectsIP

4.2.2 **Usuarios sospechosos por contenido del curso**

A lo largo de este sub-apartado explicamos el desarrollo llevado a cabo para implementar el segundo criterio comentado al principio del apartado. Se trata de identificar a aquellos usuarios que han realizado los problemas del curso sin haber accedido en ningún momento al contenido de éste, ya sea bien por vídeo o por documento PDF. Estas acciones son bastante sospechosas, ya que para realizar correctamente los problemas propuestos durante el curso se debe acceder al contenido subido a la plataforma.

Para identificar los usuarios que cumplen este criterio, primero filtraremos aquellos usuarios que no contienen ningún evento de vídeo o documento durante todo su transcurso realizando el curso y guardaremos el resultado en un dataframe. Esta consulta se puede ver en la Figura 17.

```
val selectQuery: String =
    s"""
    SELECT id_usuario as id_usuario_docs,0 as num_docs_iter
    FROM $events_view
    WHERE id_usuario not in
        ( SELECT distinct id_usuario
        FROM
            (SELECT id_usuario, count(*) as num_docs_iter
            FROM $events_view
            WHERE upper(evento) = upper('textbook.pdf.chapter.navigated')
            or upper(evento) like "%VIDEO%"
            GROUP BY id_usuario)
        t_docs )
    GROUP BY id_usuario
    """
```

Figura 17 - Consulta para seleccionar los usuarios que no contienen eventos relacionados con vídeos o documentos.

De manera similar, guardaremos en otro dataframe el resultado de todos aquellos usuarios que tienen uno o más eventos de problemas donde sus respuestas a los ejercicios sean correctas, como se puede visualizar en la Figura 18.

```
val df_probs_iter = spark.sql(s"SELECT id_usuario as id_usuario_probs,count(*) as num_probs_iter" +
    s" FROM $events_view, $results_view " +
    s" WHERE upper(evento) = upper('problem_check') and $events_view.id_evento = $results_view.id_evento "+
    s" and upper(correcto) = upper('True') "+
    s" GROUP BY id_usuario")
```

Figura 18- Consulta para seleccionar los usuarios que tienen 1 o más eventos de problemas realizados correctamente.

Finalmente, seleccionaremos únicamente aquellos usuarios que se encuentren en ambos dataframes, es decir, que cumplan el caso de no tener ningún evento relacionado con el contenido del curso y a su vez haber respondido correctamente a algún problema. Este último paso puede verse implementado en la Figura 19.

```
val probs_without_docs = df_docs_iter.join(df_probs_iter,
                         df_probs_iter("id_usuario_probs") === df_docs_iter("id_usuario_docs"),
                         "inner")
```

Figura 19 – Cruce de dataframes calculados por id_usuario

4.2.3 **Usuarios sospechosos por timestamp**

En este sub-apartado detectaremos aquellos pares de usuarios que cumplen con el tercer criterio especificado. Éstos serán sospechosos de haber cometido algún tipo de engaño a lo largo del curso por haber realizado un mismo ejercicio dentro del mismo problema con un intervalo menor de 30 segundos con otro usuario distinto y, o bien haber contestado lo mismo o que el usuario con timestamp mayor haya contestado correctamente tras haber fallado el usuario que realizó antes el ejercicio, habiendo podido obtener la respuesta al final.

La consulta SQL que utilizaremos para detectar a estos pares de usuarios se puede visualizar en la Figura 20.

```
val selectQuery: String =
 s"""
 WITH users_probs AS
     (SELECT distinct id_usuario, $events_view.id_evento, CAST(tiempo as timestamp) as timestamp,
 id_problema, correcto, id_ejercicio, respuesta
 FROM $events_view, $results_view
 WHERE upper(evento) = upper('problem_check') and $events_view .id_evento = $results_view.id_evento),
 t_user1 AS
     (select id_usuario as id_usuario1, id_evento as id_evento1, timestamp as timestamp1,
     id_problema as id_problema1,correcto as correcto1, id_ejercicio as id_ejercicio1,
     respuesta as respuesta1
     FROM users_probs),
 t_user2 AS
     (SELECT id_usuario as id_usuario2, id_evento as id_evento2, timestamp as timestamp2,
     id_problema as id_problema2, correcto as correcto2, id_ejercicio as id_ejercicio2,
     respuesta as respuesta2
     FROM users_probs)
 SELECT id_usuario1, id_usuario2
 FROM
     (SELECT id_usuario1, id_evento1, timestamp1, id_problema1, correcto1, id_ejercicio1, respuesta1,
     id_usuario2, id_evento2, timestamp2, id_problema2, correcto2,  id_ejercicio2, respuesta2,
     CAST(timestamp1 as long) - CAST(timestamp2 as long) as diffTime
     FROM t_user1
     INNER JOIN t_user2
     WHERE t_user1.id_usuario1 > t_user2.id_usuario2) t_aux
 WHERE
 (diffTime < 0 and diffTime > -30 and id_problema1 = id_problema2 and id_ejercicio1 = id_ejercicio2
 and (respuesta1 = respuesta2 OR (upper(correcto1) = upper('False') AND upper(correcto2) = upper('True') ) ) )
 OR
 (diffTime < 30 and diffTime >=0 and id_problema1 = id_problema2 and id_ejercicio1 = id_ejercicio2
 and (respuesta1 = respuesta2 OR (upper(correcto2) = upper('False') AND upper(correcto1) = upper('True') ) ) )
 """
```

Figura 20 - Consulta para seleccionar pares de usuarios que cumplen el criterio 3

Se puede ver que es una query bastante parecida a la utilizada en sub-apartado de usuarios sospechosos por compartir la misma IP. Sin embargo, cabe destacar que en esta consulta no filtramos en ningún momento comparando los valores del campo IP.

Asimismo, sí que filtraremos por un rango de intervalo en el timestamp para que el tiempo siempre sea menor de 30 segundos. Utilizamos este intervalo de tiempo tan reducido para identificar aquellos pares de usuarios que son posibles sospechosos de haber realizado juntos los problemas.

Por último, al igual que en el primer criterio implementado, agruparemos el resultado final de los pares de usuarios por su identificador y cuantificaremos cuántas veces un mismo usuario ha sido seleccionado como sospechoso según el tercer criterio, almacenando el resultado en un nuevo campo dentro del dataframe y creando una vista temporal de éste llamada table_suspectsTime, como podemos ver en la Figura 21, que será utilizada más adelante.

```
val df_suspect_diffTime = spark.sql(s"$selectQuery")

val count_users2 = df_suspect_diffTime.groupBy("id_usuario2")
                    .count()
                    .withColumnRenamed("count","count_users2")

val count_users1 = df_suspect_diffTime.groupBy("id_usuario1")
                    .count()
                    .withColumnRenamed("count","count_users1")

var users_num_iter_suspectTime = count_users2.alias("df_count_users2")
                                .join(count_users1.alias("df_count_users1"),
                                    $"df_count_users1.id_usuario1" === $"df_count_users2.id_usuario2","outer")
                                .withColumn("count_users2",
                                    when(col("count_users2").isNull,lit(0)).otherwise(col("count_users2")))
                                .withColumn("count_users1",
                                    when(col("count_users1").isNull,lit(0)).otherwise(col("count_users1")))
                                .withColumn("sum_iters",
                                    List(col("count_users1"), col("count_users2")).reduce(_+_))

users_num_iter_suspectTime.createOrReplaceTempView("table_suspectsTime")
```

Figura 21 - Creación de la vista temporal table_suspectsTime

4.3 Clasificación y modificación del dataset

Tras haber obtenido los resultados comentados en los sub-apartados anteriores, clasificaremos como usuarios sospechosos aquellos que hayan cumplido 2 o más de los criterios determinados, por lo que los que únicamente cumplan un criterio no serán clasificados como tal.

Para clasificarlos, contamos con un dataset desarrollado antes de comenzar con este proyecto y facilitado por mi tutora. Este dataset cuenta con un registro por cada usuario perteneciente al curso y 50 atributos que calculan diferentes características de los eventos vinculados con el curso que ha llevado a cabo, como por ejemplo; número total de eventos realizados, número total de interacciones con problemas, etc. Al analizar este conjunto de datos, vimos que ninguno de los 50 atributos estaba relacionado con un campo que contase la cantidad de eventos donde realiza un problema que comparte IP con otro usuario distinto y otro campo que determinase la cantidad de eventos donde realiza un problema en un intervalo menor de 30 segundos con otro usuario y contestando de manera similar.

Como hemos comentado en los anteriores sub-apartados, estos dos cálculos los almacenamos en dos vistas temporales al calcular el primer criterio y el tercero, por lo que decidimos crear un nuevo conjunto de datos añadiendo estos dos nuevos atributos para cada usuario. Del segundo criterio tomamos la determinación de no añadir ningún atributo ya que contábamos con uno que indicaba el número total de interacciones con problemas, y otros dos que indicaban el número total de interacciones con documentos y con vídeos, respectivamente. Por lo que finalmente, contamos con el dataset original clasificado según los criterios que hemos determinado, y con un nuevo dataset con dos atributos añadidos, pero con la misma clasificación.

Para clasificar cualquiera de los datasets tendremos en cuenta que éste está construido formato CSV delimitando cada campo con ";" y con una cabecera que contiene el título de cada atributo. En la Figura 22, podemos ver cómo hemos procedido a realizar la lectura de este fichero.

```
var csv_file = spark.read.format("csv")
  .option("header", "true").option("delimiter", ";")
  .load(s"$originalDataset")
  .withColumnRenamed("id_usuario","id_usuario_csv")
```

Figura 22 – Lectura del fichero CSV en Spark

Tras haber almacenado en un dataframe todo el conjunto de datos, cruzaremos con el dataframe donde tenemos aquellos usuarios que determinamos que eran sospechosos y añadiremos un nuevo campo que estará a 1 cuando el cruce por *id_usuario* se haya dado, y a 0 para aquellos usuarios que no se encuentran en el dataframe de sospechosos, como se puede ver en la Figura 23.

```
csv_file  = csv_file.join(users_suspect,
    users_suspect("id_usuario") === csv_file("id_usuario_csv"), "left")
  .withColumn("class",
     when(col("id_usuario").isNull, lit(0)).otherwise(lit(1)))
  .drop("id_usuario")
```

Figura 23 – Cruce de dataframes para clasificar los usuarios fraudulentos del dataset

Finalmente, guardamos el resultado del dataset clasificado en un fichero CSV con cabecera y delimitador ";", obteniendo un conjunto de datos clasificado según los criterios propuestos al principio de este apartado. Este paso puede visualizarse en la Figura 24.

```
csv_file.select(reorder.head, reorder.tail: _*).coalesce(1)
  .write.format("csv")
  .option("header", "true")
  .option("delimiter", ";")
  .save(s"$newDataset")
```

Figura 24 – Escritura de fichero CSV en Spark

5 Predicción y análisis de los resultados

A lo largo de este apartado, estudiaremos la posibilidad de poder predecir usuarios fraudulentos, según los criterios comentados en el apartado anterior, a partir de las interacciones realizadas a lo largo de todo el curso. Cada una de éstas, ya sea con documentos, vídeos, actividades, etc. pertenecerán como una característica del usuario dentro de nuestro dataset, compuesto por 50 atributos distintos.

Durante este análisis, clasificaremos con diferentes modelos, comparando sus resultados para determinar qué modelo sería el óptimo en nuestro trabajo, y analizaremos los resultados de distintas métricas utilizadas. Por último, estudiaremos cómo afectan distintas técnicas de balanceo a este dataset y veremos si realizando ciertas modificaciones conseguimos aumentar el rendimiento de nuestro clasificador elegido.

En este primer apartado cargaremos el dataset original clasificado según nuestros criterios para determinar si un usuario es sospechoso de fraude o no. Hemos optado por una división de datos aleatoria para el train y test, con un 85% y un 15% de los datos, respectivamente. Utilizaremos cuatro modelos diferentes de clasificación para comparar resultados entre ellos y determinar cuáles serán las mejores opciones para predecir la clasificación en este proyecto.

5.1 Definición de los modelos

En este sub-apartado, antes de comenzar con el análisis de los resultados de los diferentes modelos de clasificación, realizaremos una breve descripción del funcionamiento de cada uno de ellos para comprender mejor cómo funciona internamente cada predicción.

El primer modelo de clasificación utilizado es Naive Bayes. Este modelo es un clasificador probabilístico basado en el Teorema de Bayes, tras aplicar algunas hipótesis simplificadoras. Éste asume que las características son independientes entre sí dada la clase, lo que puede expresarse como:

$$P(C, X_1, X_2, \dots, X_n) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

Donde C es la clase y X_i las características. Asimismo, Naive Bayes combina este modelo probabilístico con la regla de decisión MAP (máximo a posteriori), consiguiendo tomar como resultado del clasificador la hipótesis más probable de las calculadas

$$argmax_c P(C = c|X_1, X_2, \dots, X_n)$$

El segundo modelo utilizado para este apartado es Regresión Logística. Éste es un modelo lineal en función de las variables independientes para dar la probabilidad de la clase, siendo su formulación equivalente:

$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i})}}$$

Esta fórmula también es comúnmente conocida como un perceptrón simple, es decir, una red neuronal de una sola capa. La siguiente ecuación representaría el cálculo realizado por una red neuronal simple, por lo que podemos apreciar un parecido bastante razonable a la fórmula anterior:

$$y = \frac{1}{1 + e^{-f(x)}}$$

La finalidad de implementar dichas fórmulas sigmoidales, es conseguir obtener la probabilidad de una variable, es decir, de obtener un valor entre "0" y "1" dado el conjunto de características. Cabe destacar que la librería Sklearn nos proporciona una serie de hiperparámetros para conseguir mejorar el ajuste de nuestro modelo, tales como:

- *Solver*: Parámetro el cual indicará el algoritmo a seguir por el modelo para realizar la clasificación. En nuestro caso, escogimos '*liblinear*', para indicar que realice una clasificación lineal.
- *C:* Parámetro de regularización que ayuda al control del sobreajuste cuanto más bajo es su valor, por lo que nuestra implementación le indicamos el valor de '*2e-2*'.
- *Class_weight:* Parámetro para indicar si estás tratando con un conjunto de datos desbalanceado o no. En nuestra implementación le indicamos el valor de '*balanced*' para indicar que trabajaremos con un conjunto desbalanceado.
- *Warm_start:* Parámetro utilizado para reducir el tiempo de procesamiento del Kernel en los casos en los que trabajamos con conjunto de datos muy pesados. El valor asignado en nuestro desarrollo fue '*True*', ya que contamos con una volumetría grande de datos.

El tercer modelo utilizado es SVM (Support Vector Machine). Una SVM parte de un espacio de dimensionalidad muy alta donde construye o bien un hiperplano, o un conjunto de éstos. Gracias al espacio de trabajo del modelo, podemos aplicar este método a problemas de clasificación o regresión. SVM es capaz de predecir, a partir de un conjunto de puntos clasificados en dos posibles categorías, a qué categoría pertenecerán cada uno de los puntos de un nuevo conjunto, al igual que el resto de clasificadores mencionados.

Asimismo, a este tipo de clasificadores también se les conoce como "de margen máximo" debido a su funcionalidad. Este tipo de algoritmos busca un hiperplano de separación basándose en los puntos más cerca de él mismo, intentando maximizar la distancia, es decir, que los puntos más cercanos tengan el máximo margen. Con esta funcionalidad implementada, conseguimos dividir los puntos en diferentes categorías utilizando el hiperplano como limitador.

Al igual que en el caso de Regresión Logística, Sklearn también nos proporciona un conjunto de hiperparámetros para ajustar debidamente nuestro modelo SVM, como por ejemplo:

- *Kernel*: Es el hiperparámetro principal dentro del modelo SVM. Dependiendo del valor que se le asigne aplicará una función de kernel distinta a la hora de clasificar los puntos dentro del hiperplano. En nuestro proyecto, hemos escogido el kernel que aplica la función de base radial, es decir, '*rbf*'.
- *C*: En el caso de SVM, obtuvimos los mejores resultados indicando el valor '*2e-2*'.

- *Gamma*: Este hiperparámetro es utiliza únicamente en el kernel que hemos seleccionado. Indicaría la influencia que tiene dentro de nuestro modelo cada registro de entrenamiento, que en nuestro caso hemos seleccionado que el valor de influencia sea escalado ('*scale*'), dependiendo del peso de cada ejemplo.
- *Class_weight*: Al igual que en el caso de Regresión Logística, asignamos el valor de '*balanced*' para indicar que trabajaremos con un conjunto desbalanceado.
- *Random_state*: Este parámetro indicará la semilla para la generación de valores pseudoaleatorios dentro de la inicialización de ciertos parámetros del modelo. En nuestro caso, seleccionamos el valor de '*0*' para indicar a nuestro modelo que los valores seleccionados de manera aleatoria no se cambien en ningún momento.
- *Probability*: Parámetro al que le asignamos el valor '*True*' para permitir calcular estimaciones de probabilidad en nuestro modelo.

El último modelo utilizado es Random Forest [8]. Este modelo se presenta como una mejora de la técnica utilizada en lo árboles de decisión. Esta técnica se basa en llegar a combinar un conjunto de árboles de decisión aleatorizados entrenados sobre un conjunto de datos. Para clasificar un conjunto nuevo de datos cada árbol evaluará el dato de entrada y la predicción final del modelo será la clase más votada por los árboles.

Los árboles son los candidatos esenciales debido a que pueden llegar a registrar complejas interacciones entre los datos y pueden crecer suficientemente profundo. Por último, indicaremos los hiperparámetros que nos ofrece Sklearn en el caso del modelo Random Forest, y que hemos hecho uso de ellos:

- *N_estimators*: Parámetro que indica el número de árboles que va a tener nuestro modelo. Durante nuestra implementación, indicamos el valor de 10.
- *N_jobs*: Parámetro que indica el número de cores que se pueden usar para entrenar los árboles. Le asignamos el valor de -1 para indicar que utilice todos los recursos disponibles.
- *Max_depth*: Parámetro que indica la profundidad máxima de un árbol. Indicamos el valor de 5 durante nuestro desarrollo.

En el Anexo B, podemos observar el código Python implementado para llevar a cabo este apartado. Tras distintas pruebas de predicción realizadas, comprobamos que los mejores resultados los obteníamos con los hiperparámetros tomando los valores comentados anteriormente.

5.2 Análisis de los resultados

Tras entrenar a cada uno de nuestros modelos, a la hora de predecir los datos test, obtenemos los siguientes resultados. Comenzaremos con el análisis de los resultados del modelo de Naive Bayes que se pueden observar en la Figura 25.

	precision	recall	f1-score	support
0.0	0.95	0.16	0.27	372
1.0	0.21	0.97	0.34	86
accuracy			0.31	458
macro avg	0.58	0.56	0.31	458
weighted avg	0.81	0.31	0.28	458

Porcentaje de acierto del modelo Naive Bayes: 0.30786

Figura 25 – Resultados de las técnicas de evaluación de Naive Bayes

Observando únicamente el porcentaje de acierto, podemos determinar que no es un clasificador bueno para nuestro dataset, ya que sólo obtenemos un 30,79% de acierto a la hora de predecir la clasificación de nuestros datos. Si analizamos los resultados de la clase negativa (No fraudulento) y la clase positiva (Fraudulento), podemos visualizar que el número de ejemplos de cada clase está bastante desbalanceado.

La clase negativa tiene un total de 372 registros de test, mientras que, la clase positiva contaría únicamente con 86. Analizando los datos obtenidos de la clase negativa, podemos observar que nuestro modelo tiene un porcentaje alto de Precision, mientras que muy bajo Recall. Esto puede significar que a la hora de clasificar como clase negativa los ejemplos suele predecirlo correctamente, sin embargo, muchos ejemplos de clase negativa no están siendo predichos como tal. Si nos fijamos en los resultados obtenidos en la clase minoritaria, podemos ver que ocurre lo inverso a la clase anterior, es decir, obtiene muy bajo porcentaje de Precision y muy alto Recall. Estos datos nos llevan a deducir, que seguramente nuestro modelo esté clasificando como positivos muchos ejemplos, de los cuales muchos no son de esa clase, lo que explicaría los resultados de la clase mayoritaria.

En la Figura 26, podemos ver los resultados del modelo de Regresión Logística.

	precision	recall	f1-score	support
0.0	0.92	0.70	0.79	372
1.0	0.36	0.72	0.48	86
accuracy			0.71	458
macro avg	0.64	0.71	0.64	458
weighted avg	0.81	0.71	0.74	458

Porcentaje de acierto del modelo Regresión Logística: 0.70524

Figura 26 - Resultados de las técnicas de evaluación de Regresión Logística

En este caso, nuestro modelo ha obtenido un porcentaje de acierto mayor para nuestro dataset que Naive Bayes, con un total de 70,52%. Analizando la clase negativa vemos que ha obtenido mejores resultados a la hora de predecirla, pero sin embargo, en el caso de la clase positiva sigue teniendo unos resultados muy bajos. Podemos observar que aunque la Precision ha mejorado con un 0.36, en el caso de Recall ha bajado comparado con Naive Bayes a un 0.72. Esto quiere decir que aunque nuestro modelo de Regresión Logística no esté clasificando tantos registros como positivos cuando en realidad eran negativos, la

clasificación de los verdaderos positivos ha empeorado provocando que aumente el número de registros con una clasificación negativa cuando en realidad formaban parte de la clase positiva dentro de nuestro conjunto de datos.

En la Figura 27, podemos ver los resultados del modelo de SVM.

	precision	recall	f1-score	support
0.0	0.88	0.69	0.77	372
1.0	0.30	0.58	0.40	86
accuracy			0.67	458
macro avg	0.59	0.63	0.58	458
weighted avg	0.77	0.67	0.70	458

Porcentaje de acierto del modelo SVM: 0.66594

Figura 27 - Resultados de las técnicas de evaluación de SVM

Si comparamos el resultado del SVM y del clasificador anterior (Regresión Logística), los resultados son muy parecidos. Porcentaje de acierto de 66,59%, junto con unos valores de aceptables para la clase negativa pero para la clase positiva muy bajos.

En la Figura 28, podemos ver los resultados del modelo de Random Forest.

	precision	recall	f1-score	support
0.0	0.88	0.98	0.93	372
1.0	0.82	0.42	0.55	86
accuracy			0.87	458
macro avg	0.85	0.70	0.74	458
weighted avg	0.87	0.87	0.86	458

Porcentaje de acierto del modelo Random Forest: 0.87336

Figura 28 - Resultados de las técnicas de evaluación de Random Forest

En este último caso, podemos ver que el porcentaje de acierto es el más alto de todos con un 87,34% de acierto, además de obtener muy buenos valores para la clase negativa. Aunque la clase positiva haya obtenido mejores resultados en el cálculo de las métricas de rendimiento que el resto de ejemplos de modelos entrenados, siguen siendo valores bastante bajos (Precision = 0.82, Recall = 0.42, F1-Score = 0.55). Esto nos indica, que dentro de los modelos utilizados, claramente el Random Forest sería la mejor opción pero sigue sin ser suficientemente bueno para conseguir predecir el fraude dentro de nuestro dataset.

Como hemos podido ver anteriormente, mirar únicamente el resultado del porcentaje de acierto de un modelo no es suficiente como para determinar si es un buen modelo o no para nuestro caso específico de predicción.

Ya hemos analizado las principales métricas de rendimiento (Recall, Precisión, F1-Score), y ahora veremos de dónde proceden esos resultados. Calcularemos la matriz de confusión de los 4 modelos entrenados y analizaremos todos los valores (Verdaderos Positivos,

Verdaderos Negativos, Falsos Positivos, Falsos Negativos). Estos valores son los que se utilizan para calcular las métricas de rendimiento mencionadas anteriormente y nos indica cuántos de los datos que están clasificados como 0, el modelo ha predicho que será 0, y cuántos usuarios clasificados como 0, el modelo se ha equivocado y ha predicho que serán 1, y viceversa. Estos resultados no serán de gran ayuda para poder ver si, efectivamente las deducciones mencionadas anteriormente, se cumplen. En la Figura 29, podemos ver el resultado de la matriz de confusión de cada uno de los modelos. Dentro del cuadrado dividido en 4 secciones, cabe indicar que: La sección de arriba-izquierda serían los Verdaderos Negativos, la sección de arriba-derecha serían los Falsos Positivos, la sección de abajo-izquierda serían los Falsos Negativos, y la sección de abajo-derecha son los Verdaderos Positivos.

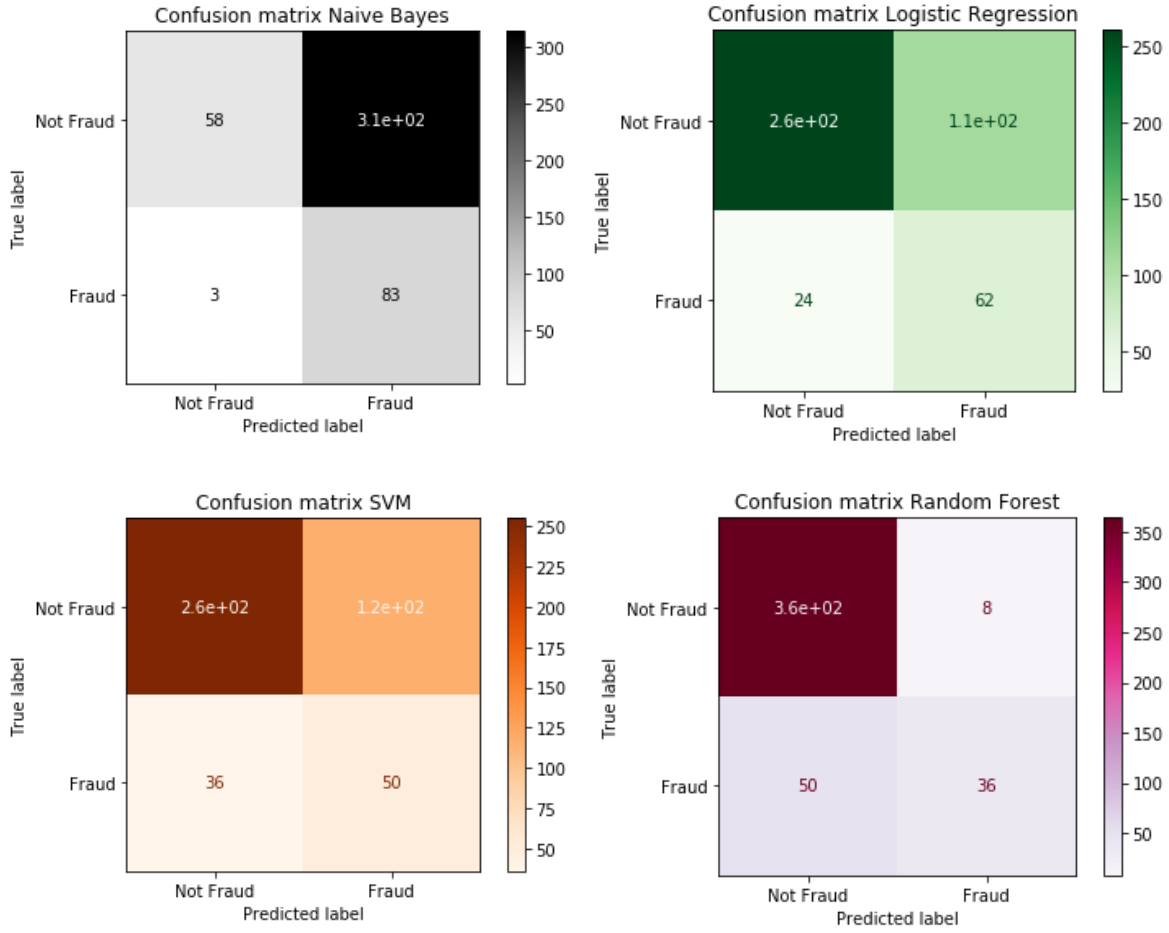


Figura 29 – Matrices de confusión de los cuatro modelos

El mejor de los resultados posibles sería obtener tanto en los Falsos Positivos, como en los Falsos Negativos un total de 0. Dependiendo del tipo de clasificación, normalmente uno de los casos sería el más crítico. Más adelante, analizaremos cuál de los dos será en el que nos enfocaremos en nuestro caso específico para obtener el menor de los resultados posibles.

Comenzaremos analizando las matrices de los modelos con el mismo orden que en anteriores ocasiones, es decir, empezando por los resultados obtenidos del modelo Naive Bayes. Podemos ver que prácticamente ha clasificado todo nuestro conjunto de datos como positivo

ya que hemos obtenido FP=310 y TP=83, lo que hace un total de 393 registros de los 458 de nuestro conjunto de datos test. Si observamos el caso de Regresión Logística y SVM (los cuales tienen resultados bastante parecidos), podemos ver que el número de FP llega a ser de 110 y 120, respectivamente. Por lo tanto, vemos que en estos dos casos claramente ha bajado el número de falsos positivos predichos con estos modelos y aumentado el número de verdaderos negativos comparado con el anterior modelo.

Sin embargo, estos resultados no siguen siendo del todo buenos ya que, si sumamos por cada modelo los FP y FN obtenemos un total de 134 y 156 registros mal clasificados, lo que equivale a un 29,26% y 34,06% del total de conjuntos utilizados en la predicción (que concuerdan con el 70,52% y 66,59% de acierto de cada modelo mencionado al principio del apartado).

Finalmente, obtenemos la matriz de confusión del clasificador Random Forest, la cual obtiene FN = 50 y FP = 8, que hasta ahora son los valores más bajos. A pesar de ello, los Falsos Negativos siguen siendo un número bastante alto, ya que será nuestro punto crítico de error, el cual intentaremos minimizar al máximo posible en los siguientes apartados.

Para finalizar con el análisis de los cuatro modelos utilizados, calcularemos la curva ROC y compararemos los resultados obtenidos. Esta curva compara el porcentaje de los Verdaderos Positivos en función de los Falsos Positivos del modelo entrenado, y se puede visualizar en la Figura 30. El AUC Score es el cálculo que aparece en la leyenda de la gráfica abajo a la izquierda y oscila entre [0.5 - 1]. Cuanto mayor sea el valor, mejor habrá sido la predicción de nuestro clasificador.

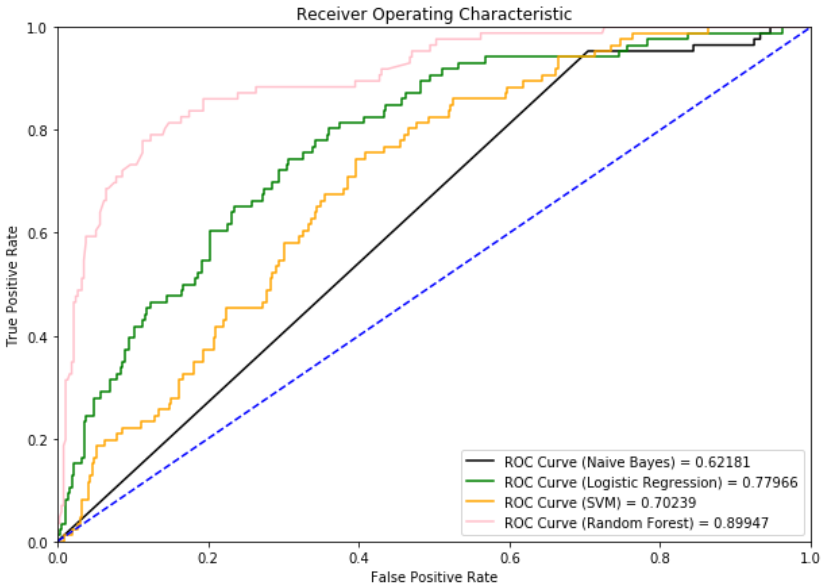


Figura 30 – Curva ROC de los cuatro modelos

Podemos apreciar que el valor más bajo obtenido es en el caso del modelo Naive Bayes con un AUC score de 0.622. Los modelos de Regresión Logística y SVM oscilan entre un 0.7-0.78, lo que indican un AUC score aceptable. Pero si nos centramos en el caso específico del modelo Random Forest, vemos que alcanza un AUC=0.89947, es decir, un resultado bastante bueno para la clasificación. Analizando todos los resultados obtenidos de cada uno

de nuestros modelos, podemos deducir que aunque en ciertos casos los resultados no llegan a ser del todo malos, pueden mejorarse. Por este motivo, vamos a centrarnos en los próximos análisis en intentar mejorar el peor de los modelos con técnicas de balanceo de datos y comprobar cómo afecta al rendimiento de éste a la hora de predecir, y por el contrario ver si conseguimos mejorar el modelo con los mejores resultados obtenidos hasta ahora y concluir si es posible implementar un modelo capaz de predecir nuestra clasificación de manera eficiente.

A raíz de estos primeros resultados, hemos podido ver que unos de los principales problemas es nuestro dataset original es el desbalanceo de ejemplos de datos entre las clases, por lo que a continuación en la Figura 31, mostraremos un histograma con los dos tipos de clases y el número de registros por cada uno de ellos.

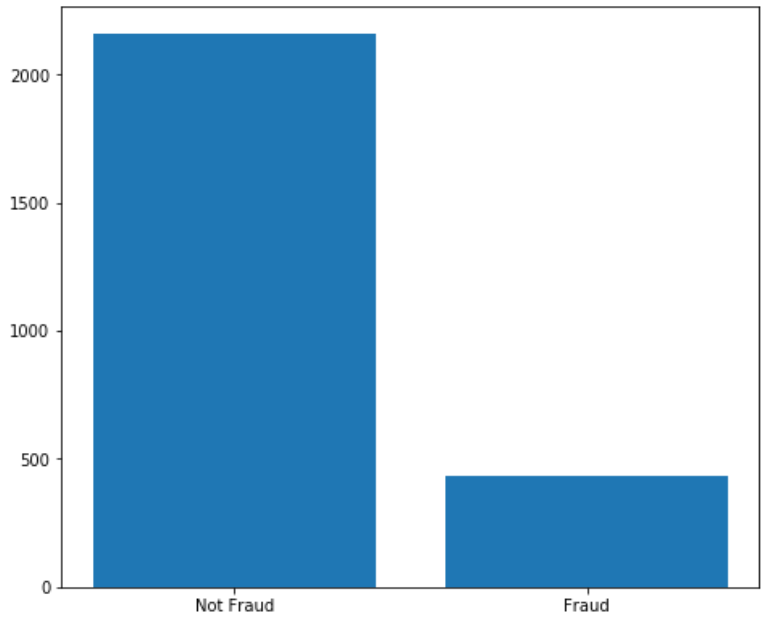


Figura 31 – Histograma del número de registros clasificados como fraudulentos o como no fraudulentos

Podemos observar, una clara diferencia entre ambas clases. Estos datos que hemos utilizado son únicamente los de entrenamiento ya que son los que nos interesan modificar y balancear para conseguir un entrenamiento de los modelos más equitativo. Vemos que de la clase negativa contamos con 2160 usuarios, mientras que de la clase positiva sólo obtenemos un total de 434 (menos de un 17% del total de datos). A partir de este análisis, aplicaremos para los datos de entrenamiento técnicas de balanceo.

Las técnicas más conocidas para balancear un conjunto de datos clasificados son Oversampling y Undersampling. El Oversampling trata de aumentar el número de registros de la clase minoritaria hasta que sea equivalente al de la clase mayoritaria. Hay distintas técnicas para realizar esta creación de nuevos datos, pero nosotros utilizaremos únicamente RandomOverSampler para ver de manera sencilla cómo puede afectar a los resultados. RandomOverSampler consta de ir atribuyendo nuevos valores (que ya se encontraban dentro de la clase minoritaria), a los atributos de manera aleatoria creando así nuevos datos. Por otro lado, el Undersampling consta de eliminar registros de la clase mayoritaria hasta que

contenga el mismo número de ejemplos que la clase minoritaria. En nuestro caso utilizaremos también la técnica RandomUnderSampler.

RandomUnderSampler utiliza una técnica, al igual que RandomOverSampler, aleatoria escogiendo datos de nuestra clase mayoritaria y eliminándolos hasta equiparar el número de ejemplos de cada clase, por lo que se nos quedaría un total de 434 ejemplos tanto para la clase positiva como para la negativa, mientras que con RandomOverSampler serían 2160 ejemplos para la clase positiva y negativa. Podemos apreciar, que en el caso de la técnica UnderSampling contamos con la desventaja de pérdida de información de nuestro conjunto de datos, al haber eliminado 1726 ejemplos en la clase mayoritaria. Al ser Naive Bayes el modelo con peores resultados en los análisis anteriores, lo utilizaremos para analizar cómo puede afectar las técnicas de balanceo en el entrenamiento de un clasificador.

En la Figura 32, podemos observar el resultado de las matrices de confusión del modelo Naive Bayes tras aplicar Undersampling (matriz de la izquierda) y Oversampling (matriz de la derecha).

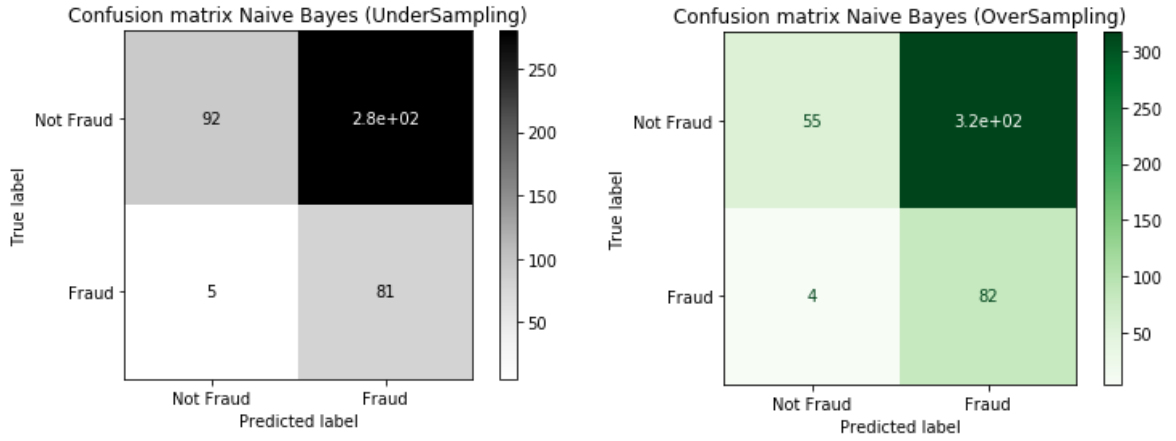


Figura 32 - Matrices de confusión del modelo Naive Bayes aplicando distintas técnicas de balanceo.

Al analizar estos datos, podemos ver que ambos clasificadores lo que están realizando es una clasificación única de todos los datos como fraudulentos prácticamente. Nuestro clasificador ha predicho correctamente 81-82 usuarios como fraudulentos, respectivamente. Por otro lado, alrededor de 280-320 usuarios, los clasificó como clase positiva, siendo en realidad éstos de la clase negativa. En conclusión, ha clasificado de todo nuestros datos de tipo test únicamente como no fraudulentos (92+5)-(55+4) de un total de 458 ejemplos de usuarios siendo 372 no fraudulentos. Por último, compararemos resultados de la curva ROC tanto del primer modelo entrenado con nuestro dataset original, como con los dos modelos en los que se ha aplicado estas dos técnicas de balanceo en la Figura 33.

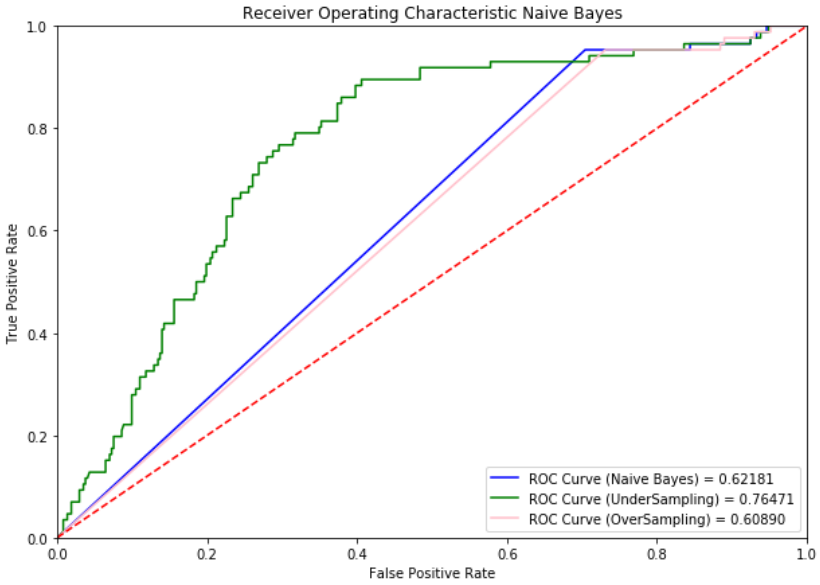


Figura 33 – Curvas ROC del modelo Naive Bayes aplicando distintas técnicas de balanceo y sin balanceo

Como vemos en la gráfica, los resultados son bastante similares, siendo el mejor de ellos el modelo que utilizó la técnica de balanceo Undersampling. El modelo entrenado con el dataset original y el modelo entrenado con la técnica aplicada de Oversampling sobre los datos de entrenamiento, han obtenido resultados prácticamente idénticos en la curva ROC.
Tras este breve análisis de ambas técnicas, podemos determinar que pueden mejorar la predicción de nuestro modelo a la hora de aplicarlas para realizar el entrenamiento, pero depende mucho del conjunto de datos que estemos tratando ya que en nuestro caso podemos observar que ha mejorado los resultados la técnica UnderSampling con la cual eliminamos información del dataset perteneciente a la clase negativa.

El modelo de Random Forest, que fue el modelo con mejor resultado, consta de una función que balancea automáticamente los datos a la hora del entrenamiento, solucionando este problema. A continuación, utilizaremos este clasificador balanceado de Random Forest y analizaremos los resultados de la Figura 34.

	precision	recall	f1-score	support
0.0	0.96	0.83	0.89	372
1.0	0.53	0.84	0.65	86
accuracy			0.83	458
macro avg	0.74	0.83	0.77	458
weighted avg	0.88	0.83	0.84	458

Porcentaje de acierto del modelo Random Forest: 0.82969

Figura 34 - Resultados de las técnicas de evaluación de Random Forest balanceado

Vemos que los resultados son muy similares al primer clasificador del Random Forest utilizado sin balancear los datos. Tras estos resultados, vemos que el principal problema no debe de estar únicamente en el desbalanceo de nuestro dataset, por lo que vamos a estudiar los atributos de nuestro conjunto de datos. Como comentamos al principio, nuestro dataset consta de 50 atributos para clasificar a cada usuario. Cada uno de estos atributos tiene un porcentaje de importancia para el modelo Random Forest a la hora de predecir la clase, como podemos apreciar en la Figura 35, donde aparecen los 10 atributos más relevantes de nuestro conjunto de datos.

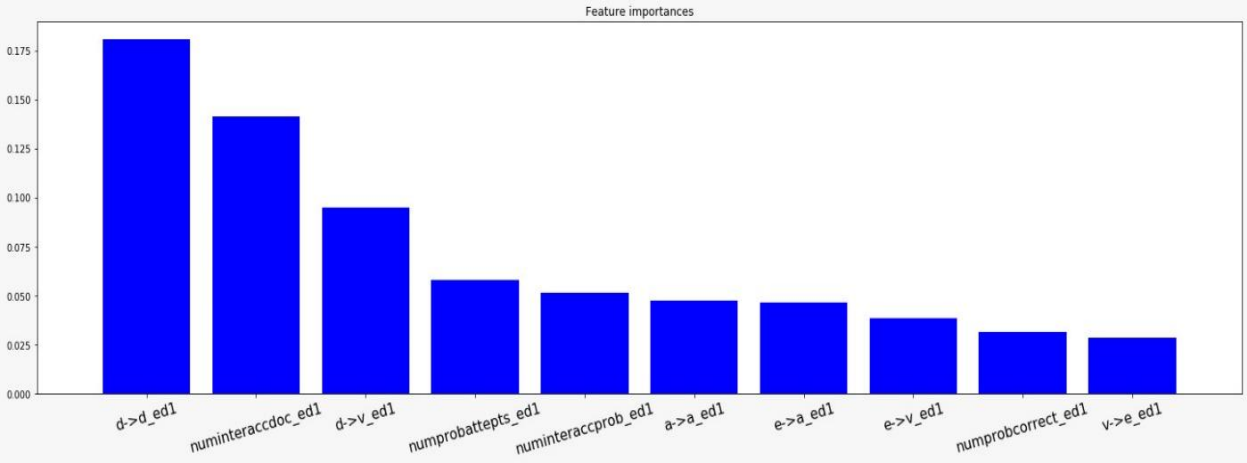


Figura 35 – Histograma con los diez atributos más relevantes para nuestro modelo Random Forest

Vemos en el histograma que muchos de los atributos de nuestro dataset tienen poca relevancia a la hora de clasificar nuestros usuarios, ya que teniendo sólo en cuenta los diez primeros atributos, podemos apreciar que el décimo tiene una relevancia aproximada de 0.025, lo que significa que los restantes 40 atributos tendrán una relevancia inferior a ésta. Al final de la Sección 4, explicamos la implementación que llevamos a cabo para modificar y generar un nuevo conjunto de datos con dos atributos añadidos mediante un proceso Spark. Asimismo, eliminaremos los atributos menos relevantes del conjunto y volveremos a analizar los resultados de ese nuevo modelo, comparando los resultados de nuestro modelo Random Forest utilizando cada uno de los datasets mencionados.

Los dos nuevos atributos que hemos añadido a este dataset son los siguientes:

- Número de interacciones en las que el usuario ha compartido la misma IP que otro usuario distinto.
- Número de interacciones en las que un usuario ha contestado igual que otro usuario distinto al mismo problema en un intervalo inferior a 30 segundos.

Los siguientes resultados, que se pueden visualizar en la Figura 36, son de un nuevo modelo Random Forest balanceado con estos dos nuevos atributos para cada usuario.

	precision	recall	f1-score	support
0.0	0.97	0.95	0.96	377
1.0	0.80	0.88	0.84	81
accuracy			0.94	458
macro avg	0.89	0.91	0.90	458
weighted avg	0.94	0.94	0.94	458

Porcentaje de acierto del modelo Random Forest Agregrando nuevos atributos al dataset: 0.93886

Figura 36 - Resultados de las técnicas de evaluación de Random Forest balanceado con nuevos atributos en el dataset

Vemos una clara mejoría en los resultados con este total de 52 atributos. Con un 93,39% de acierto en la predicción y con resultados en las métricas calculadas para ambas clases superior a 0.8. Si volvemos a visualizar los 10 atributos más relevantes para nuestro modelo contando con estos dos nuevos que hemos añadido, podemos apreciar en la Figura 37, que se encuentran entre los tres atributos más influyentes en la clasificación.

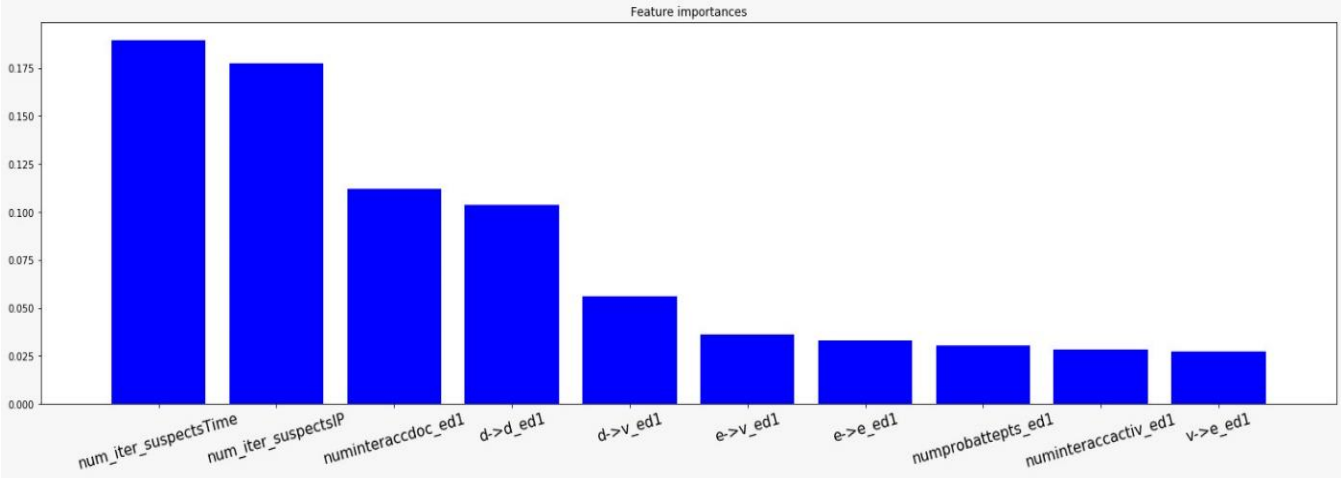


Figura 37 - Histograma con los diez atributos más relevantes para nuestro modelo trabajando con los dos nuevos atributos del dataset

Tras obtener estos buenos resultados, probaremos a eliminar los atributos que no superen el umbral estimado de importancia en un 0.025. Por lo tanto, finalmente nos quedaríamos con un total de 10 atributos, junto con la clase.

Los siguientes resultados, que podemos ver en la Figura 38, son de un nuevo modelo Random Forest balanceado entrenado con este nuevo dataset.

	precision	recall	f1-score	support
0.0	1.00	0.99	1.00	386
1.0	0.97	1.00	0.99	72
accuracy			1.00	458
macro avg	0.99	1.00	0.99	458
weighted avg	1.00	1.00	1.00	458

Porcentaje de acierto del modelo Random Forest con los atributos más relevantes del dataset: 0.99563

Figura 38 - Resultados de las técnicas de evaluación de Random Forest balanceado con atributos irrelevantes eliminados en el dataset

Podemos apreciar que tanto en la clase mayoritaria como en la minoritaria obtenemos resultados en las métricas calculadas entre 0.97-1. Además, el porcentaje de acierto es de un 99,56%, por lo que podemos confirmar que este dataset ha conseguido que nuestro modelo obtenga muy buenos resultados de predicción.

Para poder comparar los resultados de nuestros modelos Random Forest con los distintos datasets y con el balanceo de datos, calculamos sus matrices de confusión en la Figura 39.

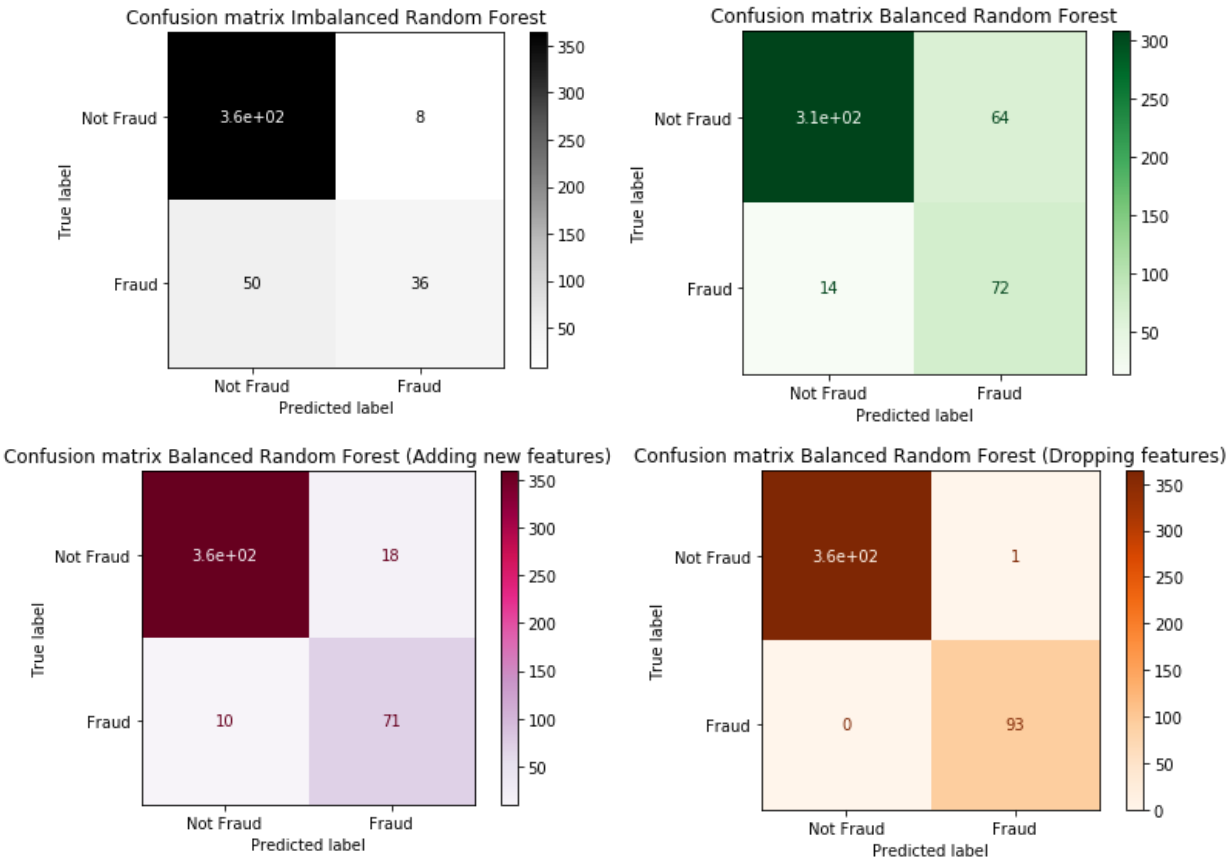


Figura 39 - Matrices de confusión del modelo Random Forest aplicando distintos datasets y técnicas de balanceos

Podemos ver un claro descenso de los falsos positivos y los falsos negativos a raíz de utilizar las técnicas de balanceo de las clases, añadir dos nuevos atributos calculados y eliminar los atributos irrelevantes en la predicción.

El mejor resultado obtenido es el Random Forest balanceado con el dataset que contiene los dos nuevos atributos y elimina los campos irrelevantes con FP = 1 y FN = 0. Este resultado es el óptimo obtenido hasta ahora, además de ser el más interesante según nuestro criterio, ya que minimizar los FN era una de nuestras prioridades y hemos conseguido que ningún usuario fraudulento sea clasificado como no fraudulento. Por último, calculamos la curva ROC de estos modelos entrenados del Random Forest en la Figura 40, para poder comparar los resultados.

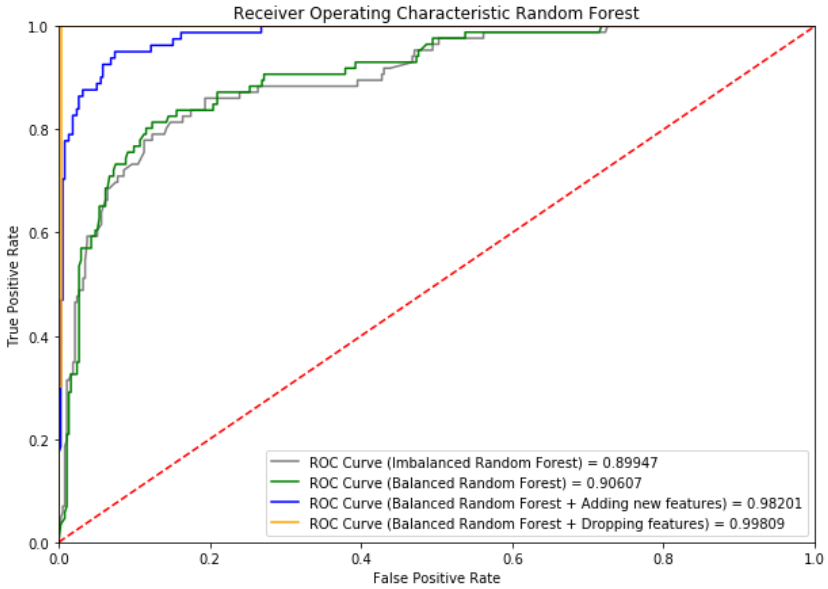


Figura 40 - Curvas ROC del modelo Random Forest balanceado aplicando distintos conjuntos de datos

Podemos ver una clara mejoría a la hora de modificar los atributos de nuestro conjunto de datos, obteniendo un AUC score = 0.99809 con nuestro modelo entrenado de Random Forest balanceado y generando un nuevo dataset mediante nuestro proceso Spark con los dos nuevos atributos y eliminando los atributos estudiados con un valor inferior de importancia de 0.025.

Por lo tanto, finalmente podemos concluir que el modelo Random Forest es el clasificador óptimo para trabajar en la predicción de nuestro dataset analizado y modificado para que no actúen ciertos atributos como atributos aleatorios a la hora de clasificar.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este trabajo final de grado hemos creado un sistema capaz de detectar usuarios fraudulentos dentro del curso online impartido por profesores de la Universidad Autónoma de Madrid en la plataforma edX. Además, también hemos estudiado la manera de poder predecir esta clasificación teniendo como objetivo principal la posible predicción de este tipo de usuarios antes de finalizar el curso.

Para poder realizar todo este trabajo de desarrollo y análisis, hemos tenido que trabajar con la tecnología Big Data para procesar los datos provenientes del sistema de logs de edX y tratar de manera eficiente grandes volúmenes de datos. Hemos tenido además, que establecer tres criterios a seguir para determinar si un usuario ha realizado algún tipo de posible fraude.

Tras el procesado, obtuvimos un dataset clasificado con todos los usuarios del curso, donde la clasificación era totalmente desbalanceada. Este problema está siendo todo un reto en el área de Data Science en los últimos años, ya que el crecimiento exponencial de los datos está dando lugar a que los datasets desbalanceados estén presentes en muchas aplicaciones reales, como en nuestro caso.

Para poder estudiar y analizar los resultados de la predicción sobre este tipo de dataset, hemos utilizado técnicas del área de Machine Learning para determinar qué algoritmo de predicción utilizar entre diferentes modelos planteados, utilizar diferentes métricas para el tratamiento de datasets desbalanceados como el sobremuestro (oversampling) o el bajomuestreo (undersampling) y analizar el resultado con diferentes medidas de evaluación del modelo entrenado.

En el caso de la predicción utilizando finalmente el algoritmo de Random Forest hemos podido observar, que tratándose de un dataset desbalanceado, no podemos evaluar la calidad de su predicción únicamente por las métricas obtenidas del porcentaje de acierto, principalmente si nos centramos en minimizar el error de los falsos negativos.

Llegamos a la conclusión de que la opción que minimizaba más este tipo de error era analizar detenidamente la importancia de cada columna del dataset para nuestro modelo seleccionado. Realizando este análisis, pudimos comprobar que contaba con muchos atributos irrelevantes a la hora de predecir la clasificación, y decidimos actualizar el dataset para eliminar los campos irrelevantes y añadir nuevos atributos a partir del proceso Big Data donde procesábamos la información de cada usuario. Con este pequeño cambio, pudimos notar una gran diferencia en los resultados de nuestro modelo a la hora de clasificar este nuevo dataset desbalanceado.

En definitiva, todo lo comentado anteriormente, demuestra que actualmente el trabajar con volúmenes tan grandes de datos es todo un reto para las empresas, las cuales cada vez almacenan mucha más información. El poder analizar estos datos se ha vuelto un problema complejo y dependiendo del error que se quiera minimizar en la clasificación será necesario utilizar técnicas totalmente distintas a la hora de entrenar el modelo.

En nuestro trabajo, hemos conseguido implementar un sistema capaz de detectar fraude en las interacciones realizadas por cada usuario dentro del curso y conseguir minimizar el error de falsos negativos obteniendo un Recall del 0.99~1.0, es decir, prediciendo correctamente como usuarios fraudulentos a los usuarios clasificados como tales en el dataset.

6.2 Trabajo futuro

El trabajo realizado de detección de fraude utilizando todas las interacciones almacenadas como logs por la plataforma edX se ha implementado muy enfocado a las características del formato generado por esta plataforma. Como trabajo futuro, podría generalizarse la detección de fraude aplicando estos criterios a cualquier sistema de logs de MOOC, para conseguir que este sistema de detección funcione para cualquier otra plataforma y curso online.

Los criterios para la detección de fraude se han determinado accediendo a toda la información disponible de cada interacción en los logs. Otra línea de trabajo futuro posible sería estudiar otros criterios diferentes con algún cierto tipo de evento no utilizado como, por ejemplo, que los foros puedan contener respuestas a exámenes y estudiar la traza de interacciones que dejan en los logs los usuarios que acceden a esos comentarios en el foro para comprobar si posteriormente en un intervalo de tiempo corto han realizado el examen que contiene esa respuesta como la correcta, o por ejemplo, estudiar los eventos relacionados con los vídeos y las interacciones de los usuarios con éstos al parar, empezar, saltar , etc.

Estudiando otros posibles criterios podríamos no sólo perfeccionar la clasificación de sospechosos a fraudulentos, sino que además podríamos comparar los resultados con los criterios determinados en nuestro trabajo y ver cuáles obtienen, posteriormente, mejores resultados a la hora de predecir su clasificación.

Para poder ejecutar este proceso Big Data tuvimos que guardar el fichero de logs en HDFS manualmente. Otro posible trabajo futuro sería poder realizar una ingesta planificada en HDFS cada vez que se generen y planificar la ejecución del proceso de la aplicación tras esta ingesta. Este tipo de trabajo futuro podría realizarse, por ejemplo con Apache Sqoop, si la ingesta se realiza desde una base de datos relacional, junto con Apache Oozie para la planificación del y ejecución del proceso a través de un flujo.

Referencias

[1] Abby Abazorius, "Study identifies new cheating method in MOOCs", MIT News Office, Abril 2015. Recuperado de http://news.mit.edu/2015/cheating-moocs-0824 [Accedido: 13-Junio-2019]

[2] Accredited Schools Online. “Preventing Plagiarism in School: Student & Teacher Resources”, 2016. Recuperado de https://www.accreditedschoolsonline.org/resources/preventing-plagiarism [Accedido: 02-Febrero-2020]

[3] Alexandron, G., Ruipérez-Valiente, J. A., & Pritchard, D. E. “Towards a general purpose anomaly detection method to identify cheaters in massive open online courses”. In Proceedings of the 12th international conference on educational data mining, (pp. 480-483), 2019.

[4] “Apache Hadoop YARN”. Recuperado de http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html [Accedido: 03-Marzo-2020]

[5] “Apache Hadoop”. Recuperado de https://hadoop.apache.org/ [Accedido: 13-Junio-2019]

[6] “Apache Spark”. Recuperado de https://spark.apache.org/ [Accedido: 13-Junio-2019]

[7] Asha, S., Chellappan, C. “Authentication of e-learners using multimodal biometric technology” International Symposium on Biometrics and Security Technologies, ISBAST, 2018.

[8] Breiman, Leo “Random Forests”, 2001. Recuperado de https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf [Accedido: 04-Marzo-2020]

[9] Calders, T., & Jaroszewicz, S. “Efficient AUC optimization for classification. In European Conference on Principles of Data Mining and Knowledge Discovery”, (pp. 42-53). Springer, Berlin, Heidelberg, Septiembre 2007.

[10] Class Central. “MOOCs by the numbers in 2018”, 2019. Recuperado de https://www.classcentral.com/report/mooc-stats-2018/ [Accedido: 22-Mayo-2019]

[11] “Dataframes y Datasets en Spark SQL”. Recuperado de https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#datasets-and-dataframes [Accedido: 13-Junio-2019]

[12] David J. Deming & Claudia Goldin & Lawrence F. Katz & Noam Yuchtman, "Can Online Learning Bend the Higher Education Cost Curve?", American Economic Review, American Economic Association, (Vol. 105(5), pp. 496-501), 2015.

[13] Dianne H.B. Welsh, Mariana Dragusin, “The New Generation of Massive Open Online Course (MOOCs) and Entrepreneurship Education”, 2013.

[14] “Ecosistema Hadoop”. Recuperado de http://spaceanalytics.blogspot.com/2016/08/introduccion-ecosistema-hadoop.html [Accedido: 13-Junio-2019]

[15] Efe. “Más de 300 detenidos por copiar masivamente en exámenes en la India”, 2015. Recuperado de http://www.antena3.com/noticias/mundo/mas-300-detenidos-copiar-masivamente-examenes-india_2015032100056.html [Accedido: 02-Febrero-2020]

[16] Foucault, M. “El examen”. Díaz Barriga. “El examen: textos para su historia y debate”. UNAM, México, 1993.

[17] Franklyn-Stokes, A., & Newstead, S. E. “Undergraduate cheating: who does what and why? Studies in higher education”, (Vol. 20(2), pp. 159-172), 1995.

[18] Fung, G. M., Mangasarian, O. L., & Shavlik, J. W. “Knowledge-based support vector machine classifiers”. In Advances in neural information processing systems (pp. 537-544), 2003.
[19] Galtung, J. “The prison; studies in institutional organization and change”. Holt, Rinehart and Winston, 1961.
[20] Gao, Q. “Biometric authentication to prevent cheating,” International Journal of Instructional Technology and Distance Learning, (Vol. 9(2), pp. 3-13), Febrero 2012.
[21] “HDFS Architecture Guide”. Recuperado de https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html [Accedido: 13-Junio-2019]
[22] Hernández, J. Á., Parra, J. M. A., & Segura, S. “El estrés ante los exámenes en los estudiantes universitarios: Propuesta de intervención. International Journal of Developmental and Educational Psychology: INFAD”. Revista de Psicología, (Vol. 2(1), pp. 55-63), 2011.
[23] Herreras, E. B. “Ansiedad ante los exámenes: evaluación e intervención psicopedagógica”. Educere, (Vol. 9(31), pp. 553-558), 2015.
[24] “Imbalanced-learn”. Recuperado de https://pypi.org/project/imbalanced-learn/ [Accedido: 13-Junio-2019]
[25] J. A. Ruipérez-Valiente, P. J. Muñoz-Merino, G. Alexandron and D. E. Pritchard, "Using Machine Learning to Detect ‘Multiple-Account’ Cheating and Analyze the Influence of Student and Problem Features," in IEEE Transactions on Learning Technologies, (Vol. 12(1), pp. 112-122), Marzo 2019.
[26] Jarrod Morgan, “How Students Cheat Online, and Why Stopping Them Matters”, Febrero 2018. Recuperado de https://www.insidehighered.com/digital-learning/views/2018/02/14/creative-cheating-online-learning-and-importance-academic [Accedido: 13-Junio-2019]
[27] Jason Brownlee, “A Gentle Introduction to k-fold Cross-Validation”, Statistical Methods, Mayo 2018.
[28] Jennifer Peterson, “An Analysis of Academic Dishonesty in Online Classes”, Mid-Western Educational Researcher, (Vol. 31(1)), 2019.
[29] L. Alvin Malesky Jr., John Baley & Robert Crow. “Academic Dishonesty: Assessing the Threat of Cheating Companies to Online Education”, College Teaching, (Vol. 6(4), pp. 178-183), 2013.
[30] Lopez-Garcia, P., Masegosa, A.D., Osaba, E. “Ensemble classification for imbalanced data based on feature space partitioning and hybrid metaheuristics”. Appl Intell 49, (pp. 2807–2822), 2019. Recuperado de https://doi.org/10.1007/s10489-019-01423-6 [Accedido: 08-Septiembre-2019]
[31] Muraskin, L. D. “Understanding Evaluation: The Way to Better Prevention Programs”, 1993.
[32] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique”, Journal of Artificial Intelligence Research, (Vol. 16, pp. 321-357), 2002.
[33] Narkhede, S. “Understanding AUC - ROC Curve”, 2019. Recuperado de https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 [Accedido: 03-Marzo-2020]
[34] Narkhede, S. “Understanding Confusion Matrix”, 2019. Recuperado de https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 [Accedido: 03-Marzo-2020]
[35] Northcutt, C. G., Ho, A. D., & Chuang, I. L. “Detecting and preventing “multiple-account” cheating in massive open online courses”. Computers & Education, 100, 71-80, Mayo 2016.

[36] Peng, M., Zhang, Q., Xing, X., Gui, T., Huang, X., Jiang, Y. G. & Chen, Z. "Trainable undersampling for class-imbalance learning". In Proceedings of the AAAI Conference on Artificial Intelligence, (Vol. 33, pp. 4707-4714), Julio 2019.
[37] Pérez-peña, R. "Students Disciplined in Harvard Scandal", 2013. Recuperado de https://www.nytimes.com/2013/02/02/education/harvard-forced-dozens-to-leave-in-cheating-scandal.html [Accedido: 02-Febrero-2020]
[38] "Prevent Plagiarism". Recuperado de https://www.turnitin.com/solutions/plagiarism-prevention [Accedido: 13-Junio-2019]
[39] "RDDs". Recuperado de https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm [Accedido: 13-Junio-2019]
[40] Rice, K. G., Choi, C. C., Zhang, Y., Morero, Y. I., & Anderson, D. "Self-critical perfectionism, acculturative stress, and depression among international students". The Counseling Psychologist, (Vol. 40(4), pp. 575-600), 2012.
[41] Robert L. Mitchell, "Pirates, cheats and IT certs", Junio 2014. Recuperado de https://www.computerworld.com.au/article/548008/pirates_cheats_it_certs/ [Accedido: 13-Junio-2019]
[42] Robert O'Brien, Hemant Ishwaran, "A random forests quantile classifier for class imbalanced data", Pattern Recognition, (Vol. 90, pp. 232-249), 2019.
[43] Rogers, R., & Neumann, C. S. "Conceptual issues and explanatory models of malingering. Malingering and illness deception", (pp. 71-82), 2003.
[44] Sara del Río, Victoria López, José Manuel Benítez, Francisco Herrera, "On the use of MapReduce for imbalanced big data using Random Forest", Information Sciences, Noviembre 2014.
[45] "Scikit-learn". Recuperado de https://scikit-learn.org/stable/ [Accedido: 13-Junio-2019]
[46] Sweeney, G. "Global corruption report: Education". Routledge, 2013.
[47] "Using Over-Sampling Techniques for Extremely Imbalanced Data", 2020. Recuperado de https://towardsdatascience.com/sampling-techniques-for-extremely-imbalanced-data-part-ii-over-sampling-d61b43bc4879 [Accedido: 03-Marzo-2020]
[48] Wright, R. E. "Logistic regression". In L. G. Grimm & P. R. Yarnold (Eds.), Reading and understanding multivariate statistics, (p. 217–244). American Psychological Association, 1995.
[49] Yiu, T. "Understanding Random Forest", 2019. Recuperado de https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accedido: 03-Marzo-2020]
[50] Zhang, H. "The optimality of naive Bayes". AA, (Vol. 1(2), pp. 3), 2004.

Anexo A – Desarrollo, compilación y ejecución del proceso Spark

A lo largo de este anexo, explicaremos con mucho más detalle el desarrollo del proceso Spark, su compilación y ejecución dentro de un clúster hadoop. El proceso ha sido implementado mediante el uso del lenguaje Scala y compilado con la herramienta Maven.

A continuación, podemos ver en la Figura 41, la estructura de nuestro repositorio de desarrollo del proyecto.

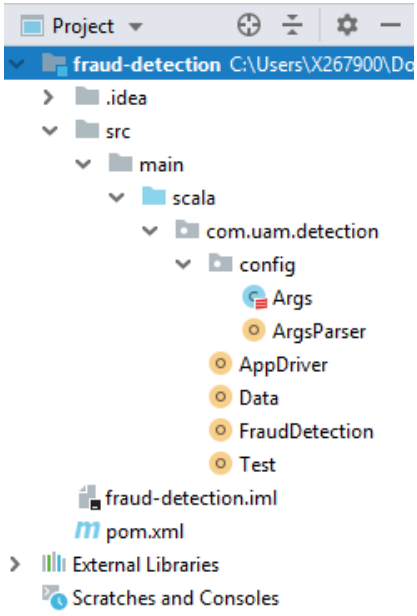


Figura 41 – Estructura del proyecto

Como podemos ver en la imagen, se trataría de un proyecto muy simple formado por un único módulo que contiene un paquete *com.uam.detection*, en el cual albergamos el driver de nuestra aplicación *AppDriver* y los objetos necesarios para la ejecución de nuestro proceso de detección de fraude según los criterios determinados. Dentro del paquete, cabe destacar la existencia de un sub-paquete *config*, el cual contaría con dos clases encargadas de la funcionalidad de parsear correctamente los argumentos pasados a nuestro módulo durante la ejecución del spark-submit.

Asimismo, podemos apreciar que nuestra aplicación contaría finalmente con un fichero XML denominado *pom*, el cual es necesario para realizar correctamente la compilación y empaquetado con la herramienta de Maven. El nombre del fichero proviene de las siglas *Project Object Model* y es utilizado principalmente para describir las dependencias, versiones, plugins, y orden de construcción de cada uno de los elementos. En el caso concreto de nuestro proyecto, contamos con la dependencia de librerías Spark (versión 2.3.0), log4j (para la edición de logs en nuestra aplicación), scopt (para el parseo de argumentos en Scala). Finalmente, contaríamos con los plugins de scala y maven shade para la construcción

de nuestro proyecto. A continuación, mostraremos la implementación del fichero pom.xml de nuestro trabajo.

Fichero pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.uam.detection</groupId>
    <artifactId>fraud-detection</artifactId>
    <version>1.0</version>

    <properties>
        <shade.extension>shaded</shade.extension>
        <spark.version>2.3.0</spark.version>
        <scopt.version>3.3.0</scopt.version>
        <config.version>1.3.1</config.version>
        <scala.binary.version>2.11</scala.binary.version>
        <java.version>1.8</java.version>
    </properties>

    <repositories>
        <repository>
            <id>scala-tools.org</id>
            <name>Scala-Tools Maven2 Repository</name>
            <url>http://scala-tools.org/repo-releases</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-hive_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <dependency>
            <groupId>com.github.scopt</groupId>
            <artifactId>scopt_2.11</artifactId>
            <version>${scopt.version}</version>
        </dependency>
    </dependencies>

    <build>
        <sourceDirectory>src/main/scala</sourceDirectory>
        <plugins>
            <plugin>
                <groupId>org.scala-tools</groupId>
                <artifactId>maven-scala-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>testCompile</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
```

```
                    <scalaVersion>${scala.binary.version}</scalaVersion>
                    <args>
                        <arg>-target:jvm-1.5</arg>
                    </args>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <finalName>${project.artifactId}-${shade.extension}</finalName>
                    <createDependencyReducedPom>false</createDependencyReducedPom>
                    <shadedArtifactAttached>true</shadedArtifactAttached>
                    <shadedClassifierName>shaded</shadedClassifierName>
                    <filters>
                        <filter>
                            <artifact>*:*</artifact>
                            <excludes>
                                <exclude>META-INF/*.SF</exclude>
                                <exclude>META-INF/*.DSA</exclude>
                                <exclude>META-INF/*.RSA</exclude>
                            </excludes>
                        </filter>
                    </filters>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

A la hora de ejecutar nuestro proyecto, llamaremos a nuestro driver que se encargará de iniciar la sesión de Spark, parsear los argumentos recibidos, iniciar la generación de logs para poder añadir éstos durante el proceso e invocar a la clase de detección de fraude. A lo largo de la siguiente implementación, se puede visualizar el fichero Scala de nuestro conector *AppDriver*.

Fichero AppDriver.scala:

```
package com.uam.detection

import com.uam.detection.config.{Args, ArgsParser}
import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object AppDriver {
  val SPARK_APP_NAME = "Fraud Detection Application"

  def main(args: Array[String]): Unit = {

    val spark = SparkSession
      .builder()
      .appName(SPARK_APP_NAME)
      .config("spark.eventLog.enabled","true")
      .getOrCreate()

    /**Generación logs en el proceso Spark*/

    val logger: Logger = Logger.getLogger("Fraud.Detection.Message")
    Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
    Logger.getLogger("org.apache.spark.storage.BlockManager").setLevel(Level.ERROR)
```

```
    logger.setLevel(Level.INFO)

    ArgsParser.parse(args, Args()).fold(ifEmpty = sys.exit(1)) { parsedArgs =>

      logger.info(s"--------Json file with IPs logs---------- ${parsedArgs.jsonIP}")
      logger.info(s"--------Json file with Videos logs---------- ${parsedArgs.jsonVid}")
      logger.info(s"---------------Dataset-------------- ${parsedArgs.originalDataset}")

      parsedArgs.job match {
        case "FraudDetection" =>
          FraudDetection.execute(spark, parsedArgs.jsonIP, parsedArgs.jsonVid,
            parsedArgs.originalDataset, parsedArgs.newDataset, logger)
        case _ =>
          logger.error(s"job parameter has an invalid value: ${parsedArgs.job}")
          sys.exit(1)
      }
    }
  }
}
```

Una vez que el conector ha invocado a nuestro proceso FraudDetection pasándole los argumentos recogidos de la ejecución, éste será el encargado de cargar correctamente los logs de nuestros ficheros JSON, aplicar los criterios determinados en la Sección 4 y guardar finalmente el resultado en un CSV con los dos nuevos atributos (también comentados y explicados en la Sección 4). Nuestro proceso sería muy simple ya que hemos modulado la funcionalidad generando un objeto para la parte de procesamiento de los datos (*Data*) y otro para la parte de aplicación de los criterios (*Test*), por lo tanto quedaría un fichero scala equivalente al siguiente.

Fichero FraudDetection.scala:

```
package com.uam.detection

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.log4j.Logger

object FraudDetection {

    def execute(spark: SparkSession, jsonIP: String, jsonVid: String,
                originalDataset: String, newDataset: String, logger: Logger): Unit = {

        val events_view: String = "events_view"
        val results_view: String = "results_view"

        Data.loadFromJson(spark, events_view, results_view, jsonVid, jsonIP, logger)

        val users_suspect: Dataset[Row] = Test.suspect_users(spark, events_view,
results view, logger)

        Data.saveIntoCSV(spark, originalDataset, newDataset, users suspect, logger)
    }

}
```

Como podemos ver, nuestro objeto *Data* contaría con las funciones de carga de los datos provenientes de los ficheros JSON en el metastore, y la creación del fichero CSV con los dos nuevos atributos. A su vez el objeto *Test* contaría con la función *suspect_users*, encargada de la aplicación de los tres criterios determinados y devolver los usuarios que cumplen los requerimientos definidos como posibles sospechosos de haber cometido algún acto fraudulento a lo largo del curso. Tanto para el fichero Scala con la implementación de *Data*, como para el fichero de *Test*, hemos ido mostrando su implementación por partes en la Sección 4, aclarando específicamente la funcionalidad de cada una de las partes. Aun así, en las siguientes páginas mostraremos los ficheros desde un punto de vista más genérico

para tener una imagen completa de ambos, aunque no entraremos en detalle de cada línea de código.

Fichero Data.scala:

```
package com.uam.detection

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.spark.sql.functions._
import org.apache.log4j.Logger

object Data {

    def loadFromJson(spark: SparkSession, events_view: String, results_view: String,
                     jsonVid: String, pathJsonIP: String, logger: Logger): Unit = {

        var resource_data = spark.read.json(s"$jsonVid")
        val resource_data_ip = spark.read.json (s"$pathJsonIP")

        resource_data = resource_data.filter(resource_data("Usuario") =!= "" &&
          resource_data("Usuario").isNotNull)

        var dfEventos = resource_data.select(resource_data("Usuario"),
            explode(resource_data("Eventos")).as("eventos"))
          .select("Usuario","eventos.*")

        dfEventos = dfEventos.withColumnRenamed("Usuario","id_usuario")
          .filter(dfEventos("evento") =!= "problem_check")
          .withColumn("ip",lit("NOT_IP"))

        var df_probsIP = resource_data_ip.select(resource_data_ip("Usuario"),
            explode(resource_data_ip("Eventos")).as("eventos"))
          .select("Usuario","eventos.*")

        df_probsIP = df_probsIP.withColumnRenamed("Usuario","id_usuario")
          .filter(df_probsIP("evento") === "problem_check")

.withColumn("currentTime",lit("NOT_VIDEO")).withColumn("id_video",lit("NOT_VIDEO"))
          .withColumn("new_speed",lit("NOT_VIDEO")).withColumn("new_time",lit("NOT_VIDEO"))
          .withColumn("old_speed",lit("NOT_VIDEO")).withColumn("old_time",lit("NOT_VIDEO"))

        val cols = dfEventos.columns
        df probsIP = df probsIP.select(cols.head, cols.tail:  *)

        dfEventos = dfEventos.union(df_probsIP)
          .withColumn("id_evento", monotonically_increasing_id()+1)

        val dfResultados = dfEventos.select(dfEventos("id_evento"),
            explode(dfEventos("resultados")).as("resultados"))
          .select("id_evento","resultados.*")
          .withColumn("id_resultado", monotonically_increasing_id()+1)


        dfEventos.drop("resultados").createOrReplaceTempView(s"$events_view")
        dfResultados.createOrReplaceTempView(s"$results_view")

    }

    def saveIntoCSV(spark: SparkSession, originalDataset: String,
                    newDataset: String, users suspect: Dataset[Row], logger: Logger): Unit
= {

        var csv_file = spark.read.format("csv")
          .option("header", "true").option("delimiter", ";")
          .load(s"$originalDataset")
          .withColumnRenamed("id_usuario","id_usuario_csv")

        val num_iter_suspectsIP = spark.sql(s"SELECT DISTINCT CASE WHEN id_usuario2 is null
```

```
THEN id_usuario1 WHEN " +
          s" id_usuario1 is null  THEN id_usuario2 ELSE id_usuario1 END as id_usuario,
sum_iters " +
          s" FROM table_suspectsip ")

        csv_file  = csv_file.join(num_iter_suspectsIP,
            num_iter_suspectsIP("id_usuario") === csv_file("id_usuario_csv"),
            "left")
          .withColumn("num_iter_suspectsIP",
              when(col("id_usuario").isNull, lit(0)).otherwise(col("sum_iters")))
          .drop("id_usuario").drop("sum_iters")

        val num_iter_suspectsTime = spark.sql(s"SELECT DISTINCT CASE WHEN id_usuario2 is
null THEN id_usuario1 " +
          s" WHEN id_usuario1 is null  THEN id_usuario2 ELSE id_usuario1 END as id_usuario,
sum_iters " +
          s" FROM table_suspectstime ")

        csv_file  = csv_file.join(num_iter_suspectsTime,
            num_iter_suspectsTime("id_usuario") === csv_file("id_usuario_csv"), "left")
          .withColumn("num_iter_suspectsTime",
              when(col("id_usuario").isNull, lit(0)).otherwise(col("sum_iters")))
          .drop("id_usuario").drop("sum_iters")

        csv_file  = csv_file.join(users_suspect,
            users_suspect("id_usuario") === csv_file("id_usuario_csv"), "left")
          .withColumn("class",
              when(col("id_usuario").isNull, lit(0)).otherwise(lit(1)))
          .drop("id_usuario")

        val cols = csv_file.columns
        val reorder = cols.filter(! _.contains("class"))
          .filter(! _.contains("id_usuario_csv")) :+ "class"

        csv_file.select(reorder.head, reorder.tail: _*).coalesce(1)
          .write.format("csv")
          .option("header", "true")
          .option("delimiter", ";")
          .save(s"$newDataset")

    }
}
```

Fichero Test.scala:

```
package com.uam.detection

import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.spark.sql.functions._

object Test {

    def suspect_users(spark: SparkSession, events_view: String, results_view: String,
logger: Logger): Dataset[Row] = {

        val result_final = executeFirst(spark, events_view, results_view, logger)
                           .union(executeSecond(spark, events_view, results_view, logger))
                           .union(executeThird(spark, events_view, results_view, logger))
                           .groupBy("id_usuario")
                           .count()
                           .withColumnRenamed("count","num_suspects")

        result_final.filter(result_final("num_suspects") >= 2).drop("num_suspects")
    }

    def executeFirst(spark: SparkSession, events_view: String, results_view: String,
logger: Logger): Dataset[Row] = {
      import spark.implicits._

        val selectQuery: String =
```

```
            s"""
            WITH users_probs AS
                (select distinct id_usuario, $events_view.id_evento, CAST(tiempo as
timestamp) as timestamp,
                id_problema, correcto, id_ejercicio, respuesta, ip
                FROM $events_view, $results_view
                WHERE upper(evento) = upper('problem_check')  and $events_view.id_evento =
$results_view.id_evento) ,
            t_user1 AS
                (select id_usuario as id_usuario1, id_evento as id_evento1,  timestamp as
timestamp1,
                id_problema as id_problema1,correcto as correcto1, id_ejercicio as
id_ejercicio1,
                respuesta as respuesta1, ip as ip1
                FROM users_probs),
            t_user2 AS
                (select id_usuario as id_usuario2, id_evento as id_evento2, timestamp as
timestamp2,
                id_problema as id_problema2, correcto as correcto2, id_ejercicio as
id_ejercicio2,
                respuesta as respuesta2, ip as ip2
                FROM users_probs)
            SELECT id_usuario1, id_usuario2
            FROM
                (SELECT id_usuario1, id_evento1, timestamp1, id_problema1, correcto1,
id_ejercicio1,
                respuesta1, ip1, id_usuario2, id_evento2, timestamp2, id_problema2,
correcto2,
                id_ejercicio2, respuesta2, ip2, CAST(timestamp1 as long) - CAST(timestamp2
as long) as diffTime
                FROM t_user1
                INNER JOIN t_user2
                WHERE t_user1.id_usuario1 > t_user2.id_usuario2 and ip1 = ip2)  t_aux
            WHERE
            ( diffTime < 0 and id_problema1 = id_problema2 and id_ejercicio1 =
id_ejercicio2  and
                (respuesta1 = respuesta2 OR (upper(correcto1) = upper('False') AND
upper(correcto2) = upper('True') ) ) )
            OR ( diffTime >=0 and id_problema1 = id_problema2 and  id_ejercicio1 =
id_ejercicio2  and
                (respuesta1 = respuesta2 OR (upper(correcto2) = upper('False') AND
upper(correcto1) = upper('True') ) ) )
            """

        val df_suspect_IP = spark.sql(s"$selectQuery")

        val count_users2 = df_suspect_IP.groupBy("id_usuario2")
                            .count()
                            .withColumnRenamed("count","count_users2")

        val count_users1 = df_suspect_IP.groupBy("id_usuario1")
                            .count()
                            .withColumnRenamed("count","count_users1")

        val users_num_iter_suspectIP = count_users2.alias("df_count_users2")
          .join(count_users1.alias("df_count_users1"),
            $"df_count_users1.id_usuario1" === $"df_count_users2.id_usuario2", "outer")
          .withColumn("count_users2",
            when(col("count_users2").isNull,lit(0)).otherwise(col("count_users2")))
          .withColumn("count_users1",
            when(col("count_users1").isNull,lit(0)).otherwise(col("count_users1")))
          .withColumn("sum_iters",
            List(col("count_users1"), col("count_users2")).reduce(_+_))

        users_num_iter_suspectIP.createOrReplaceTempView("table_suspectsIP")

        df_suspect_IP.select("id_usuario1")
            .union(df_suspect_IP.select("id_usuario2"))
            .withColumnRenamed("id_usuario1","id_usuario")
            .distinct()

    }

    def executeSecond(spark: SparkSession, events_view: String, results_view: String,
logger: Logger): Dataset[Row] = {
```

```
        val selectQuery: String =
            s"""
            SELECT id_usuario as id_usuario_docs,0 as num_docs_iter
            FROM $events_view
            WHERE id_usuario not in
                ( SELECT distinct id_usuario
                FROM
                    (SELECT id_usuario, count(*) as num_docs_iter
                    FROM $events_view
                    WHERE upper(evento) = upper('textbook.pdf.chapter.navigated')
                    or upper(evento) like "%VIDEO%"
                    GROUP BY id_usuario)
                t_docs )
            GROUP BY id_usuario
            """

        /**obtenemos los usuarios con 0 eventos de documentos*/

        val df_docs_iter = spark.sql(s"$selectQuery")

        /**obtenemos los usuarios con 1 ó más eventos de problemas*/

        val df_probs_iter = spark.sql(s"SELECT id_usuario as id_usuario_probs,count(*) as
num_probs_iter" +
            s" FROM $events_view, $results_view " +
            s" WHERE upper(evento) = upper('problem_check') and $events_view.id_evento =
$results_view.id_evento "+
            s" and upper(correcto) = upper('True') "+
            s" GROUP BY id_usuario")

        /**Realizamos un inner join cruzando por id_usuario de ambos dataframes*/


        val probs_without_docs = df_docs_iter.join(df_probs_iter,
                                    df_probs_iter("id_usuario_probs") ===
df_docs_iter("id_usuario_docs"),
                                    "inner")

        /**Devolvemos un Dataset[Row] con los id usuario que cumplen este criterio*/
        probs_without_docs.select(col("id_usuario_docs").as("id_usuario")).distinct()

    }


    /** Query que compara los tiempos de ejecución de un problema entre
      * dos usuarios distintos, si el tiempo de resolverlo es
      * menor de diffTime y las respuestas son las mismas para el mismo problema,
      * entonces esos usuarios serán sospechosos
    */
    def executeThird(spark: SparkSession, events_view: String, results_view: String,
logger: Logger): Dataset[Row] = {

      import spark.implicits._

       val selectQuery: String =
        s"""
        WITH users_probs AS
            (SELECT distinct id_usuario, $events_view.id_evento, CAST(tiempo as timestamp)
as timestamp,
        id_problema, correcto, id_ejercicio, respuesta
        FROM $events_view, $results_view
        WHERE upper(evento) = upper('problem_check') and $events_view .id_evento =
$results_view.id_evento),
        t_user1 AS
            (select id_usuario as id_usuario1, id_evento as id_evento1, timestamp as
timestamp1,
            id_problema as id_problema1,correcto as correcto1, id_ejercicio as
id_ejercicio1,
            respuesta as respuesta1
            FROM users_probs),
        t_user2 AS
            (SELECT id_usuario as id_usuario2, id_evento as id_evento2, timestamp as
timestamp2,
            id_problema as id_problema2, correcto as correcto2, id_ejercicio as
id_ejercicio2,
            respuesta as respuesta2
```

```
            FROM users_probs)
        SELECT id_usuario1, id_usuario2
        FROM
            (SELECT id_usuario1, id_evento1, timestamp1, id_problema1, correcto1,
id_ejercicio1, respuesta1,
            id_usuario2, id_evento2, timestamp2, id_problema2, correcto2,  id_ejercicio2,
respuesta2,
            CAST(timestamp1 as long) - CAST(timestamp2 as long) as diffTime
            FROM t_user1
            INNER JOIN t_user2
            WHERE t_user1.id_usuario1 > t_user2.id_usuario2) t_aux
        WHERE
        (diffTime < 0 and diffTime > -30 and id_problema1 = id_problema2 and id_ejercicio1
= id_ejercicio2
        and (respuesta1 = respuesta2 OR (upper(correcto1) = upper('False') AND
upper(correcto2) = upper('True') ) ) )
        OR
        (diffTime < 30 and diffTime >=0 and id_problema1 = id_problema2 and id_ejercicio1 =
id_ejercicio2
        and (respuesta1 = respuesta2 OR (upper(correcto2) = upper('False') AND
upper(correcto1) = upper('True') ) ) )
        """

       val df_suspect_diffTime = spark.sql(s"$selectQuery")

        val count_users2 = df_suspect_diffTime.groupBy("id_usuario2")
                            .count()
                            .withColumnRenamed("count","count_users2")

        val count_users1 = df_suspect_diffTime.groupBy("id_usuario1")
                            .count()
                            .withColumnRenamed("count","count_users1")

        var users_num_iter_suspectTime = count_users2.alias("df_count_users2")
                                        .join(count_users1.alias("df_count_users1"),
                                            $"df_count_users1.id_usuario1" ===
$"df_count_users2.id_usuario2","outer")
                                        .withColumn("count_users2",

when(col("count_users2").isNull,lit(0)).otherwise(col("count_users2")))
                                        .withColumn("count_users1",

when(col("count_users1").isNull,lit(0)).otherwise(col("count_users1")))
                                        .withColumn("sum_iters",
                                            List(col("count_users1"),
col("count_users2")).reduce(_+_))

        users_num_iter_suspectTime.createOrReplaceTempView("table_suspectsTime")

        df_suspect_diffTime.select("id_usuario1")
            .union(df_suspect_diffTime.select("id_usuario2"))
            .withColumnRenamed("id_usuario1","id_usuario").distinct()

    }


}
```

Una vez finalizada la implementación de la aplicación, pasaremos a compilar y empaquetar nuestro proyecto, generando un fichero jar para posteriormente ejecutarlo en nuestro proceso Spark.

Como podemos ver en la Figura 42, hemos seleccionado los comandos de *clean* e *install* de Maven, y la construcción de nuestro proyecto ha terminado con éxito generando dos ficheros jar: una release 1.0 y un shaded.

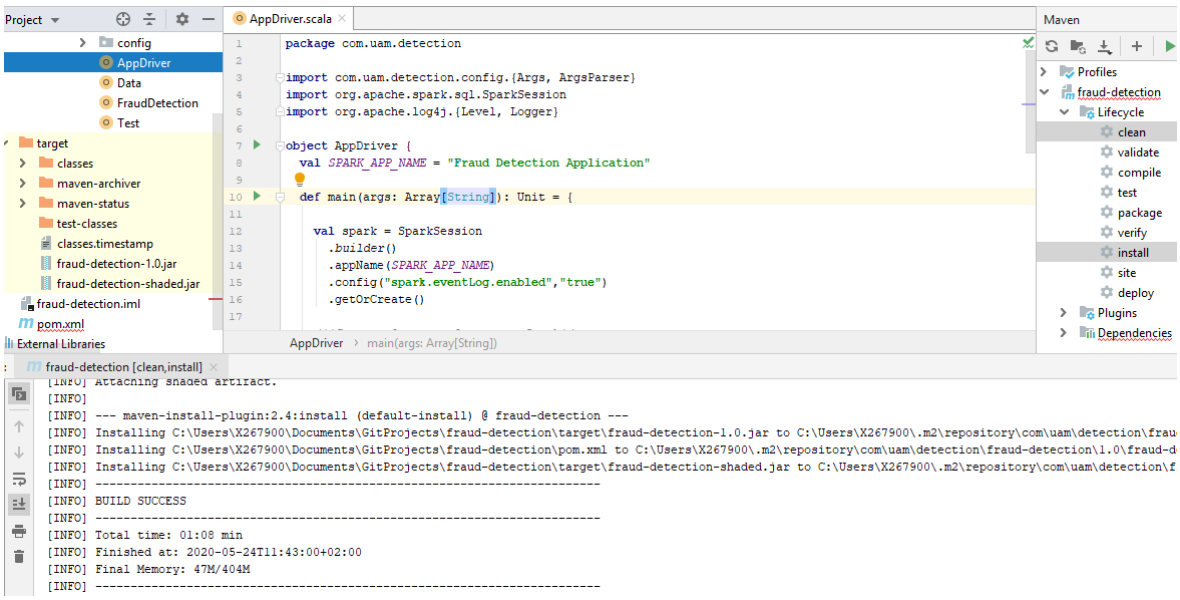


Figura 42 – Compilación y empaquetado del proyecto con Maven

Por lo que, llegados hasta este punto, ya tendríamos compilado y empaquetado nuestro proyecto para poder ejecutarlo y procesarlo.

Como hemos comentado a lo largo del trabajo, nuestro proyecto ha hecho uso de un clúster Hadoop para su ejecución. Debido a esto, hemos contado con un directorio HDFS donde almacenar los ficheros logs con los que trabajaremos, nuestro fichero jar y el dataset original del que partíamos. En la Figura 43, podemos ver nuestro directorio HDFS.

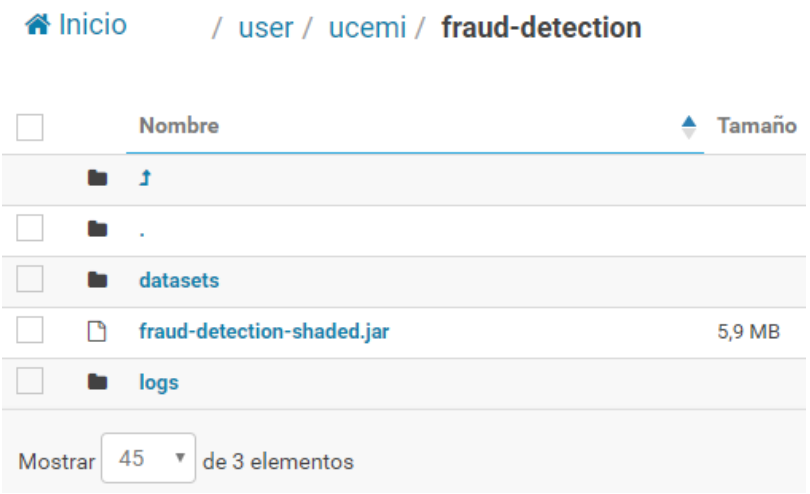


Figura 43 – Listado del directorio HDFS antes de la ejecución del proceso

Una vez tengamos listo nuestro directorio HDFS con todos los ficheros necesarios, ejecutaremos un proceso Spark mediante la estructura del spark-submit que se puede visualizar en la Figura 44.

```
spark2-submit --class com.uam.detection.AppDriver \
--master yarn \
--name "App Fraud Detection" \
--executor-cores 3 \
--num-executors 5 \
--executor-memory 1G \
--driver-memory 512M \
--conf spark.sql.shuffle.partitions=15 \
--jars hdfs:///cer/ucemi/fraud-detection/fraud-detection-shaded.jar \
--job FraudDetection \
--jsonIP hdfs:///cer/ucemi/fraud-detection/logs/eventos_conIp.json \
--jsonVid hdfs:///cer/ucemi/fraud-detection/logs/eventos_final.json \
--originalDataset hdfs:///cer/ucemi/fraud-detection/datasets/dataset_suspect.csv \
--newDataset hdfs:///cer/ucemi/fraud-detection/datasets/dataset_suspect_new_features
```

Figura 44 – Línea de comando spark-submit

Como se puede ver, ejecutaremos nuestro proceso indicando la configuración de los siguientes argumentos:

- *class*: clase main de nuestra aplicación, que en nuestro caso sería el conector.
- *master*: al indicar *yarn* indicamos que queremos que nuestro proceso se ejecute en un cluster yarn.
- *name*: nombre con el que se ejecutará nuestro proceso.
- *executor-cores*: número de cores por ejecutor.
- *num-executors*: número de ejecutores.
- *executor-memory*: memoria por ejecutor.
- *driver-memory*: memoria del driver. (por defecto, el driver es 1 core).
- *spark.sql.shuffle.partitions*: División y paralelización de tasks. (para obtener un resultado óptimo normalmente se suele indicar un número múltiplo de vcores=(ejecutor*cores))
- *jars*: fichero jar de nuestro proyecto.
- Argumentos propios de nuestro proyecto:
    - *job*: Nombre del proceso específico a ejecutar (en nuestro caso sólo contaríamos con FraudDetection).
    - *jsonIP*: ruta HDFS del fichero json con los logs IP.
    - *jsonVid*: ruta HDFS del fichero json con los logs que contienen las interacciones con los vídeos del curso.
    - *originalDataset*: ruta HDFS del dataset original.
    - *newDataset*: ruta HDFS del dataset clasificado con dos nuevos atributos que generaremos.

Tras lanzar la anterior línea de comandos de spark-submit, podemos visualizar desde el Resource Manager que nuestro proceso Spark estaría corriendo, tal y como se aprecia en la Figura 45.

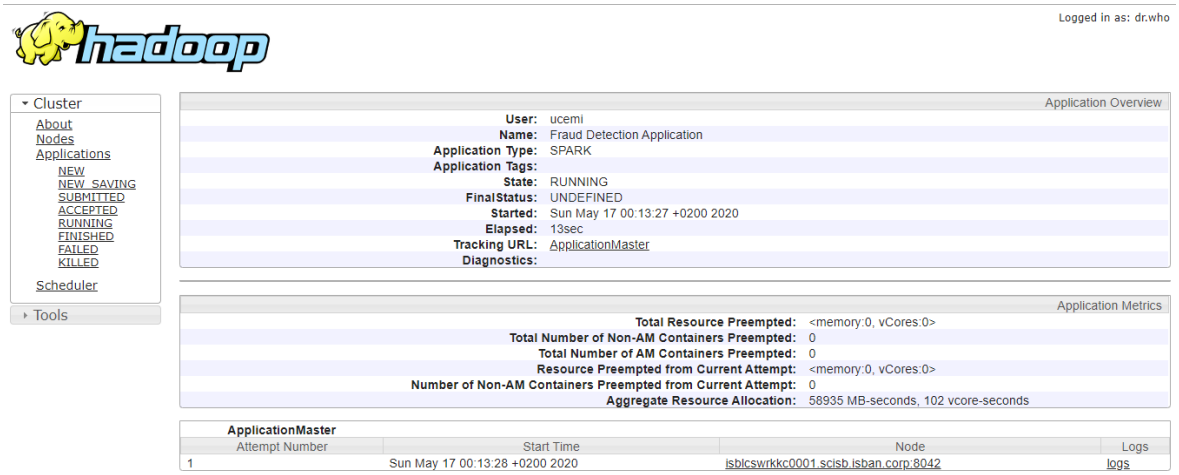


Figura 45 – Visualización del proceso Spark corriendo desde el RM

Si quisiéramos ver con más detalle la ejecución del proceso, podríamos o bien acceder a los logs que aparecen en el enlace abajo a la derecha, o acceder al enlace *ApplicationMaster* indicado en **Tracking URL**. En este último, podremos ver la traza de nuestro proceso de una manera más intuitiva y visual, como se puede ver en la Figura 46.

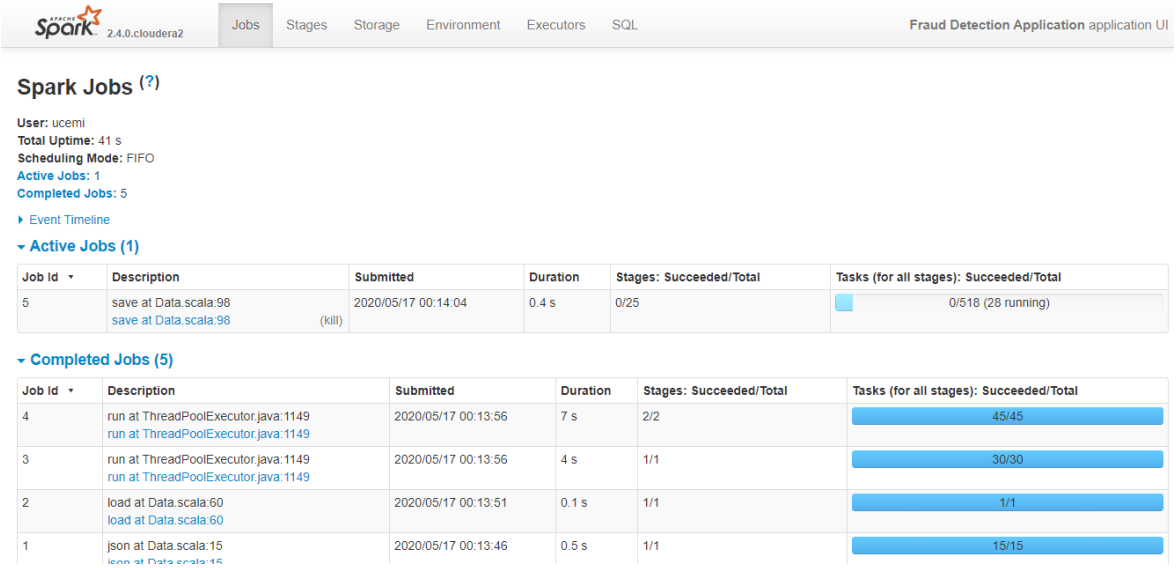


Figura 46 – Visualización de la ejecución de los jobs de nuestro proceso desde el RM

Vemos que nuestro proceso Spark ya ha llegado a ejecutar 5 jobs y se encuentra ejecutando el job que se encarga de generar nuestro nuevo dataset clasificado y almacenarlo en HDFS. Vemos que para el conjunto de stages contaría con un total de 518 tasks y actualmente se encuentran ejecutándose 28 vcores.

Si pinchamos de nuevo en el propio job, podemos incluso ver a más detalle la ejecución de éste y cada stage, como vemos en la Figura 47.

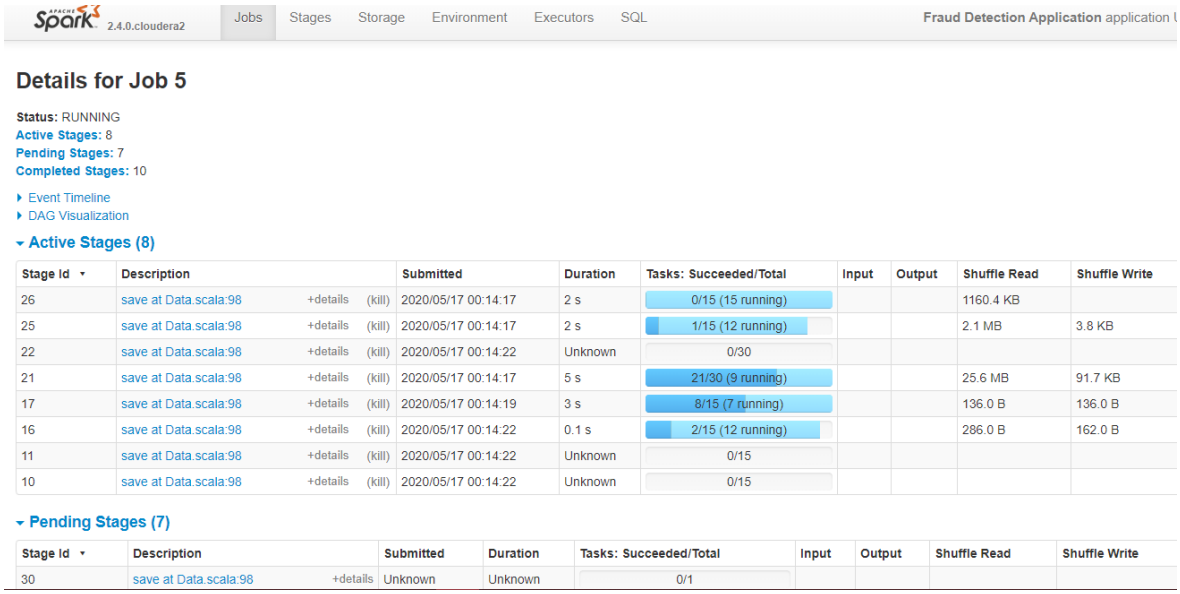


Figura 47 – Visualización de los stages en ejecución de un job desde RM

Podemos apreciar la ejecución en paralelo de varios stages de nuestro job, los cuales cuentan la mayor parte de ellos con un total de 15 tasks (como indicamos en el shuffle partitions).

Al igual que en caso de los jobs, podemos ver detalladamente la ejecución de cada stage accediendo al enlace que se encuentra en la columna **Description**. Para poder poner un ejemplo, hemos accedido al stage 1 de nuestro proceso para ver detenidamente cómo se han ejecutado las tasks con los recursos asignados a nuestro proceso. En la Figura 48 podemos ver un análisis de la ejecución llevada a cabo en este stage.

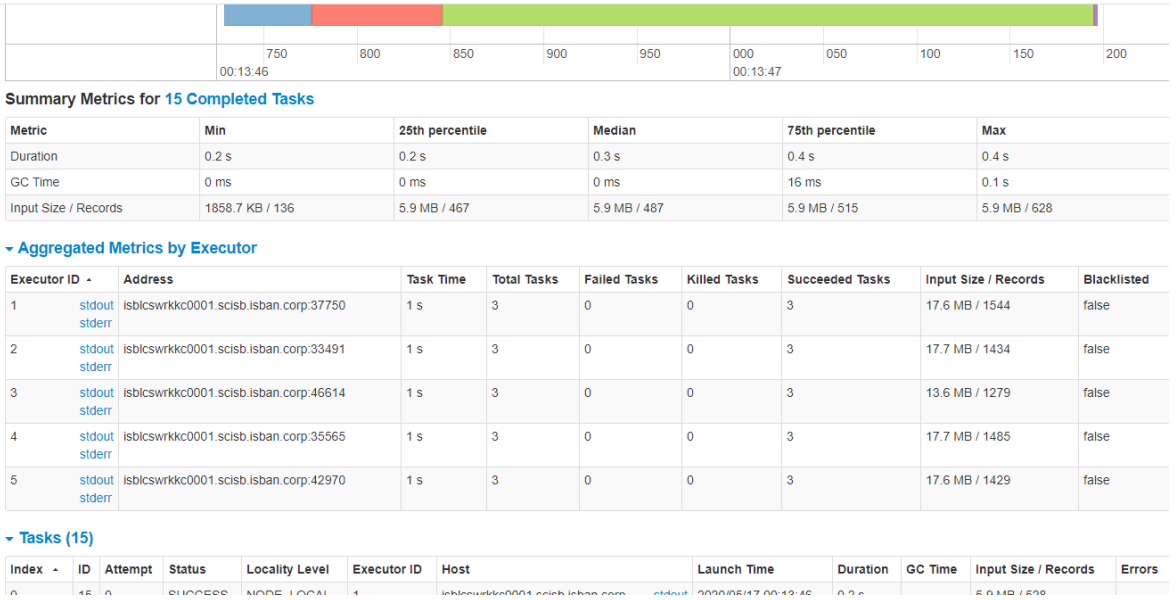


Figura 48 - Visualización de las tasks en ejecución de un stage desde RM

Contaríamos con un total de 15 tasks, como indicamos en el shuffle.partitions. Dentro del resumen de métricas de los tasks podemos apreciar que la repartición de trabajo ha sido bastante buena, ya que a partir del percentil 25 se indica una carga de trabajo de 5,9MB, la cual sería a su vez el máximo asignado a un task, y el mínimo asignado a éste sería aproximadamente 1,8MB. Vemos que la duración de cada task rondaría entre 0.2 y 0.4 segundos.

Asimismo si bajamos un poco podemos visualizar, dentro del apartado métricas agregadas por ejecutor, que nuestro stage ha contado con 5 ejecutores, ejecutándose 3 tasks en cada uno. Esto se debe a la configuración que indicamos a la hora de ejecutar nuestro proceso Spark, donde indicamos que utilizaríamos 5 ejecutores y 3 cores por ejecutor. Al contar únicamente con 15 tasks, éstas se han podido ejecutar en 5 ejecutores distintos de manera distribuida, y a su vez se han ejecutado en paralelo por cada core una tarea dentro de cada ejecutor.

Con esta configuración hemos conseguido que el tiempo de cómputo disminuya la latencia de nuestro proceso, ya que todas las tasks se habrían ejecutado en paralelo de manera distribuida sin quedar ninguna en espera de recursos.

Si abrimos el apartado de **Event Timeline**, podemos ver la ejecución de cada task en cada uno de los ejecutores a lo largo del tiempo, como se observa en la Figura 49, que cuenta con el caso de tres de los cinco ejecutores. Vemos que efectivamente se han ejecutado las tasks de manera distribuida concurrentemente.

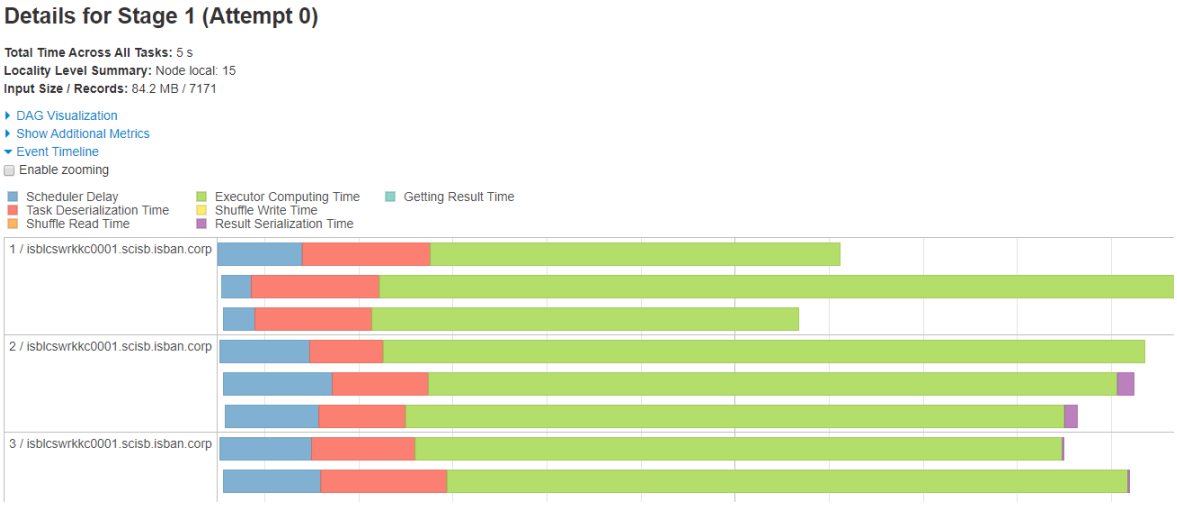


Figura 49 - Visualización de las tasks en ejecución de un stage desde RM a lo largo del tiempo

Finalmente, en la Figura 50 podemos ver que, tras un tiempo de ejecución de 1 minuto y 14 segundos, nuestro proceso Spark terminó con éxito.

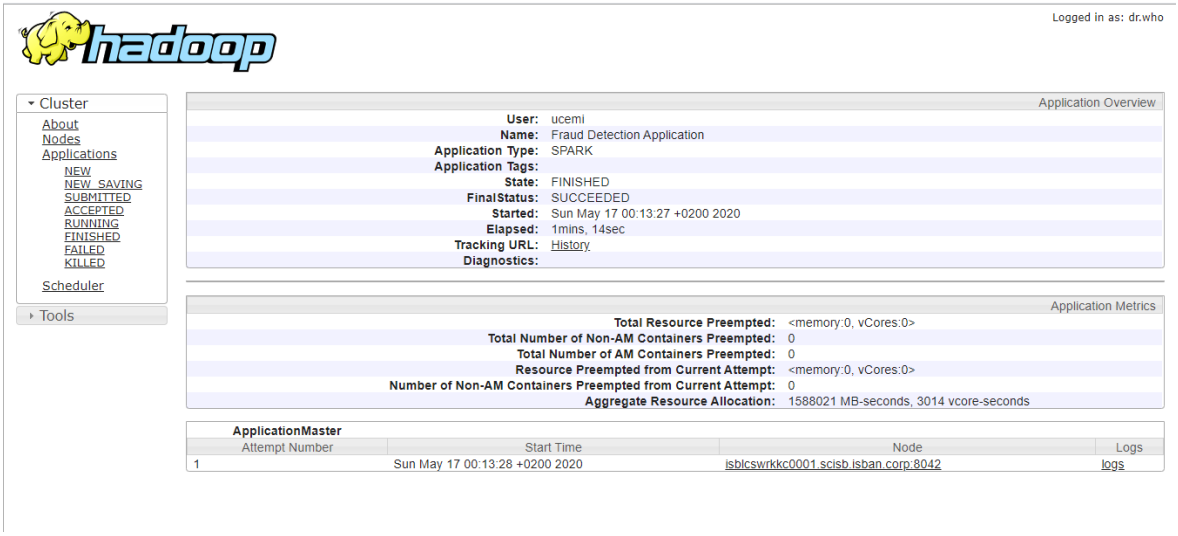


Figura 50 - Visualización del proceso Spark finalizado con éxito desde el RM

Si volvemos a listar los ficheros que se encuentran en nuestra ruta HDFS, vemos en la Figura 51 que se ha generado nuestro dataset.

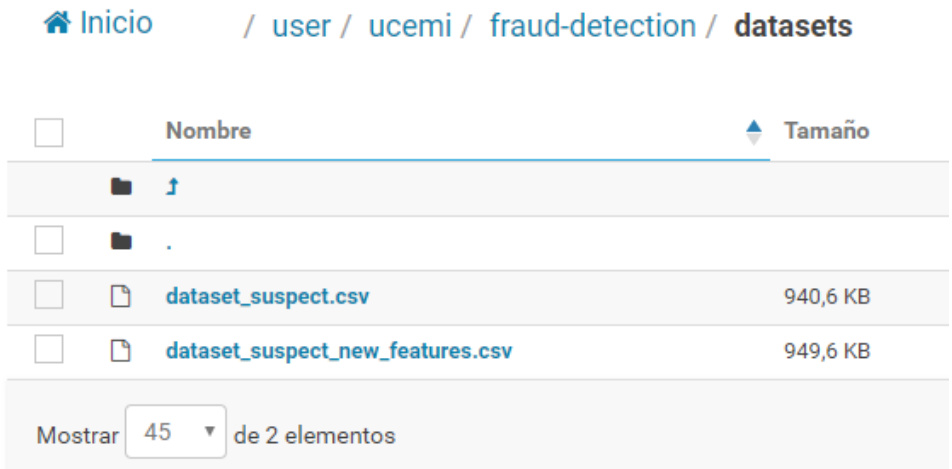
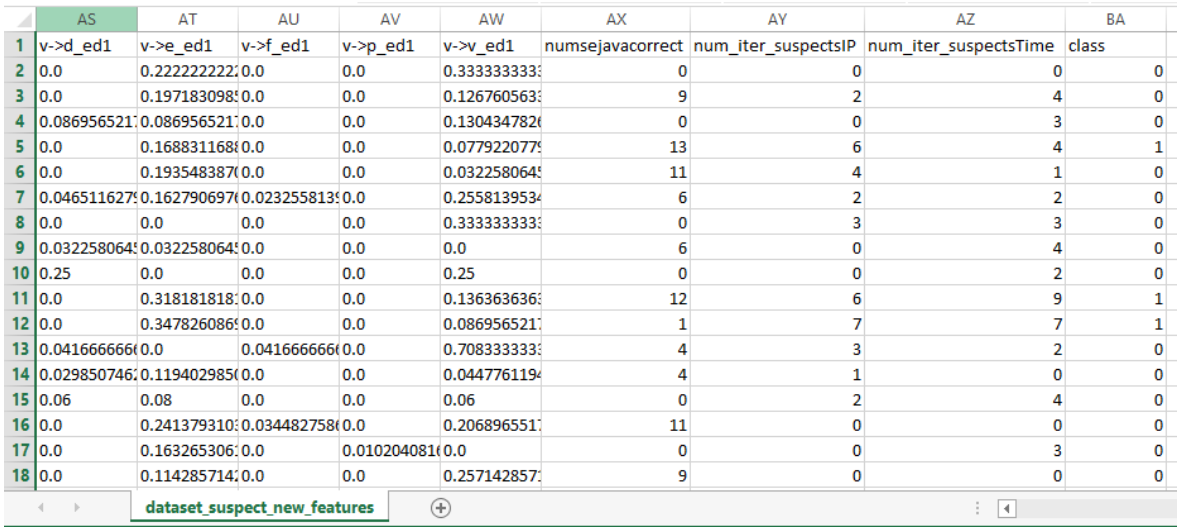


Figura 51 – Listado del directorio HDFS tras de la ejecución del proceso

El dataset con los dos nuevos atributos y la nueva clasificación, como podemos observar en la Figura 52.



	AS	AT	AU	AV	AW	AX	AY	AZ	BA
1	v->d_ed1	v->e_ed1	v->f_ed1	v->p_ed1	v->v_ed1	numsejavacorrect	num_iter_suspectsIP	num_iter_suspectsTime	class
2	0.0	0.222222222	0.0	0.0	0.333333333	0	0	0	0
3	0.0	0.197183098	0.0	0.0	0.126760563	9	2	4	0
4	0.086956521	0.086956521	0.0	0.0	0.130434782	0	0	3	0
5	0.0	0.168831168	0.0	0.0	0.077922077	13	6	4	1
6	0.0	0.193548387	0.0	0.0	0.032258064	11	4	1	0
7	0.046511627	0.162790697	0.023255813	0.0	0.255813953	6	2	2	0
8	0.0	0.0	0.0	0.0	0.333333333	0	3	3	0
9	0.032258064	0.032258064	0.0	0.0	0.0	6	0	4	0
10	0.25	0.0	0.0	0.0	0.25	0	0	2	0
11	0.0	0.318181818	0.0	0.0	0.136363636	12	6	9	1
12	0.0	0.347826086	0.0	0.0	0.086956521	1	7	7	1
13	0.041666666	0.0	0.041666666	0.0	0.708333333	4	3	2	0
14	0.029850746	0.119402985	0.0	0.0	0.044776119	4	1	0	0
15	0.06	0.08	0.0	0.0	0.06	0	2	4	0
16	0.0	0.241379310	0.034482758	0.0	0.206896551	11	0	0	0
17	0.0	0.163265306	0.0	0.010204081	0.0	0	0	3	0
18	0.0	0.114285714	0.0	0.0	0.257142857	9	0	0	0

dataset_suspect_new_features

Figura 52 – Visualización de la cabecera y los primeros registros del nuevo dataset generado

Tras haber analizado toda la parte de desarrollo y ejecución de nuestro proceso podemos llegar a la conclusión de que, en cuestión de aproximadamente 1 minuto, nuestra aplicación ha sido capaz de levantar el contexto Spark (10-20 segundos suele tardar); analizar un total de 273,2MB eliminando datos no válidos y duplicados; aplicar la detección de fraude según los criterios determinados; cargar el dataset original de 940KB; añadirle dos nuevos atributos generados, la nueva clasificación y generar un nuevo dataset de 950KB en el directorio HDFS indicado.

Anexo B – Implementación del código Python de Machine Learning

En este anexo, adjuntamos la implementación detallada de la Sección 5 de la memoria desarrollada en Python. Hemos incluido tanto los resultados como las visualizaciones y títulos, junto con comentarios para indicar a qué parte corresponde cada celda.

A continuación, mostramos el notebook realizado.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, plot_confusion_matrix,␣
↪roc_curve, auc
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression

from collections import Counter

from imblearn.ensemble import BalancedRandomForestClassifier
```

```
Using TensorFlow backend.
```

[2]:
```
# Cargamos el dataset
df = pd.read_csv("/home/blanca/Documentos/doc_TFG_actualizacion/data_suspect/
↪dataset_suspect.csv",sep=';')

data = df.values

X = data[:,:-1]
y = data[:,-1]
```

[3]:
```
# Division aleatoria del conjunto de datos entre train (0.85) y test (0.15)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)
```

0.0.1 Naive Bayes

[4]:
```
clf_nb = GaussianNB().fit(X_train,y_train)

predictions_nb = clf_nb.predict(X_test)
print(classification_report(y_test,predictions_nb))
y_pred_proba_rf_nb = clf_nb.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo Naive Bayes: %.5f'
      % clf_nb.score(X_test,y_test))

fpr_nb, tpr_nb, _ = roc_curve(y_test,y_pred_proba_rf_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.16      0.27       372
         1.0       0.21      0.97      0.34        86

    accuracy                           0.31       458
   macro avg       0.58      0.56      0.31       458
weighted avg       0.81      0.31      0.28       458

Porcentaje de acierto del modelo Naive Bayes: 0.30786
```

0.0.2 Logistic Regression

[5]:
```
clf_lr = LogisticRegression(solver='liblinear',
                            class_weight='balanced',
                            C=2e-2,
                           warm_start=True).fit(X_train,y_train)

predictions_lr = clf_lr.predict(X_test)
print(classification_report(y_test,predictions_lr))

y_pred_proba_rf_lr = clf_lr.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo Regresión Logística: %.5f'
      % clf_lr.score(X_test,y_test))

fpr_lr, tpr_lr, _ = roc_curve(y_test,y_pred_proba_rf_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.70      0.79       372
         1.0       0.36      0.72      0.48        86

    accuracy                           0.71       458
   macro avg       0.64      0.71      0.64       458
weighted avg       0.81      0.71      0.74       458

Porcentaje de acierto del modelo Regresión Logística: 0.70524
```

0.0.3 SVM

[6]:
```
clf_svm = SVC(kernel='rbf',
              gamma='scale',
              class_weight='balanced',
              C=2e-1,
              random_state=0,
              probability=True).fit(X_train,y_train)

predictions_svm = clf_svm.predict(X_test)
print(classification_report(y_test,predictions_svm))

y_pred_proba_rf_svm = clf_svm.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo SVM: %.5f'
      % clf_svm.score(X_test,y_test))

fpr_svm, tpr_svm, _ = roc_curve(y_test,y_pred_proba_rf_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.69      0.77       372
         1.0       0.30      0.58      0.40        86

    accuracy                           0.67       458
   macro avg       0.59      0.63      0.58       458
weighted avg       0.77      0.67      0.70       458

Porcentaje de acierto del modelo SVM: 0.66594
```

0.0.4 Random Forest

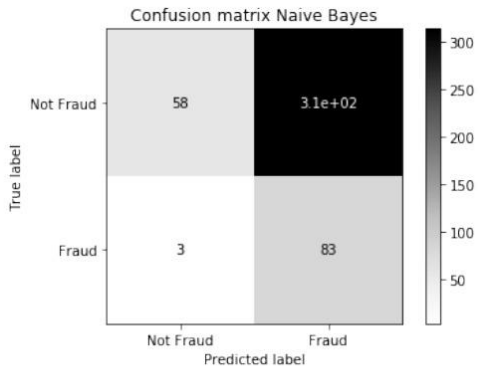
[7]:
```
clf_rf = RandomForestClassifier(n_estimators=10,
                                max_depth=5,
                               n_jobs=-1).fit(X_train,y_train)

predictions_rf = clf_rf.predict(X_test)
print(classification_report(y_test,predictions_rf))

y_pred_proba_rf = clf_rf.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo Random Forest: %.5f'
      % clf_rf.score(X_test,y_test))

fpr_rf, tpr_rf, _ = roc_curve(y_test,y_pred_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)
```

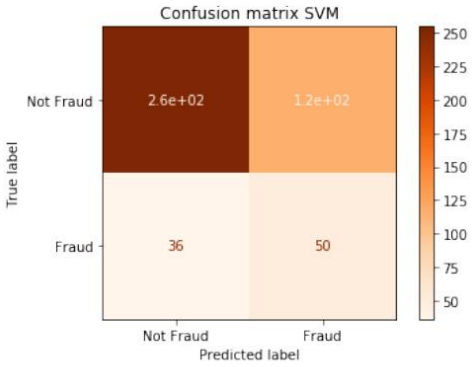
```
              precision    recall  f1-score   support
```

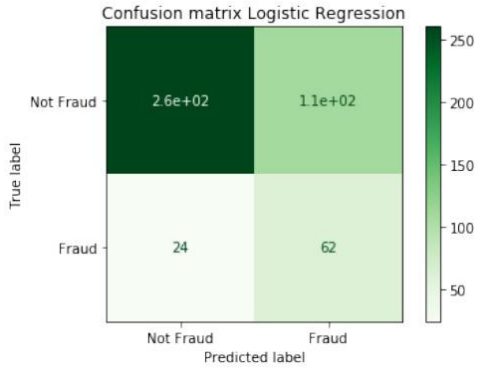
```
              0.0       0.88      0.98      0.93       372
              1.0       0.82      0.42      0.55        86

    accuracy                              0.87       458
   macro avg       0.85      0.70      0.74       458
weighted avg       0.87      0.87      0.86       458

Porcentaje de acierto del modelo Random Forest: 0.87336
```

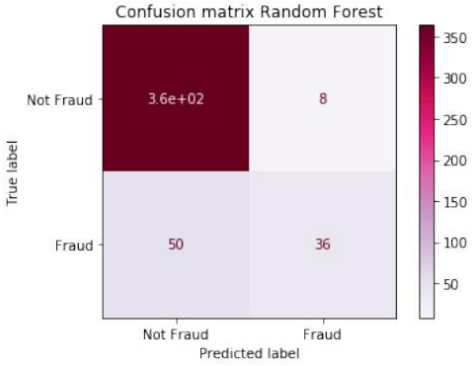
0.0.5 Matríz de confusión

[8]:
```
target_names = ['Not Fraud','Fraud']

disp_nb = plot_confusion_matrix(clf_nb, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Greys)
disp_nb.ax_.set_title('Confusion matrix Naive Bayes')

disp_lr = plot_confusion_matrix(clf_lr, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Greens)
disp_lr.ax_.set_title('Confusion matrix Logistic Regression')

disp_svm = plot_confusion_matrix(clf_svm, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Oranges)
disp_svm.ax_.set_title('Confusion matrix SVM')

disp_rf = plot_confusion_matrix(clf_rf, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.PuRd)
disp_rf.ax_.set_title('Confusion matrix Random Forest')

plt.show()
```



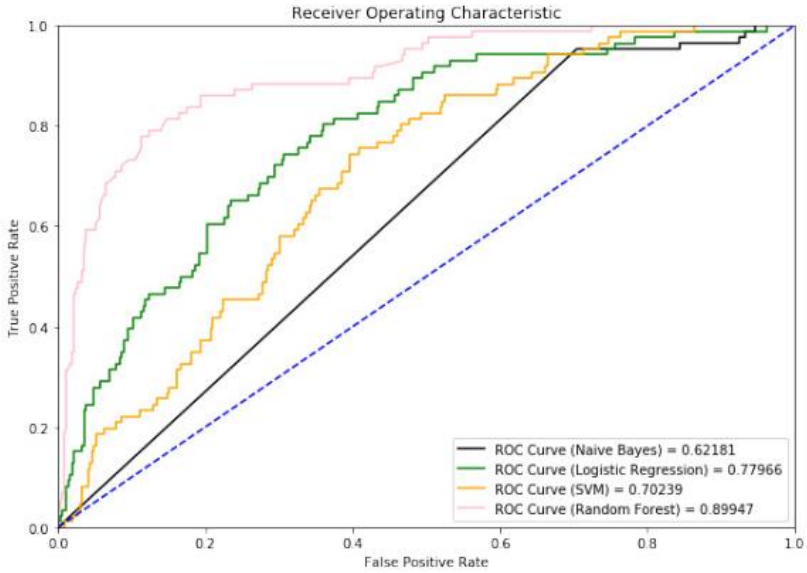




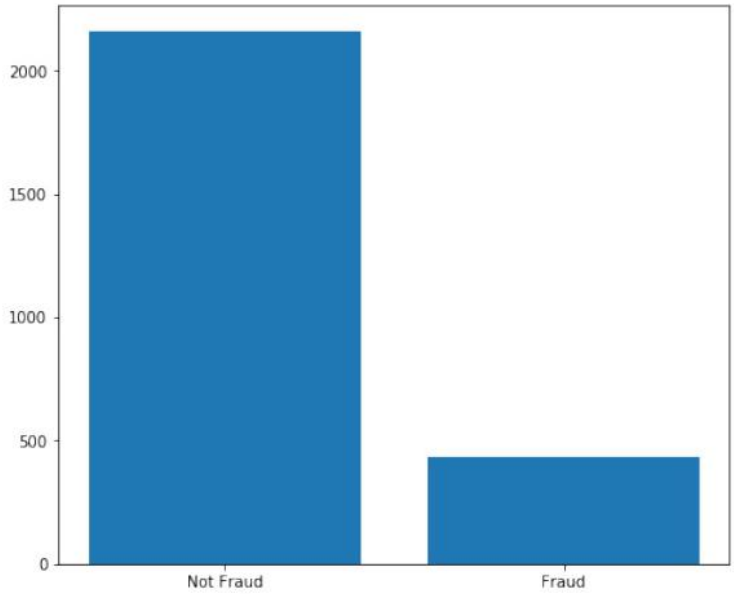


0.0.6 Curva ROC

[9]:
```
plt.figure(figsize=(10,7))
plt.title('Receiver Operating Characteristic')

plt.plot(fpr_nb, tpr_nb, 'black', label = 'ROC Curve (Naive Bayes) = %.5f' %␣
↪roc_auc_nb)
plt.plot(fpr_lr, tpr_lr, 'green', label = 'ROC Curve (Logistic Regression) = %.
↪5f' % roc_auc_lr)
plt.plot(fpr_svm, tpr_svm, 'orange', label = 'ROC Curve (SVM) = %.5f' %␣
↪roc_auc_svm)
plt.plot(fpr_rf, tpr_rf, 'pink', label = 'ROC Curve (Random Forest) = %.5f' %␣
↪roc_auc_rf)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[10]:
```
fig = plt.figure(figsize=(6,5))
ax = fig.add_axes([0,0,1,1])
langs = ['Not Fraud','Fraud']
students = np.bincount(y_train.astype('int64'))
ax.bar(langs,students)
plt.show()

print("Distribución original para el entrenamiento {}".format(Counter(y_train)))
```



```
Distribución original para el entrenamiento Counter({0.0: 2160, 1.0: 434})
```

0.1 OverSampling and UnderSampling Dataset

[11]:
```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

us = RandomUnderSampler()
os = RandomOverSampler()
X_train_us, y_train_us = us.fit_sample(X_train, y_train)
X_train_os, y_train_os = os.fit_sample(X_train, y_train)

print("Distribución después del UnderSampling para el entrenamiento {}".
 ↪format(Counter(y_train_us)))
print("Distribución después del OverSampling para el entrenamiento {}".
 ↪format(Counter(y_train_os)))
```

```
Distribución después del UnderSampling para el entrenamiento Counter({0.0: 434,
1.0: 434})
Distribución después del OverSampling para el entrenamiento Counter({1.0: 2160,
0.0: 2160})
```

0.2 Naive Bayes - UnderSampling and OverSampling

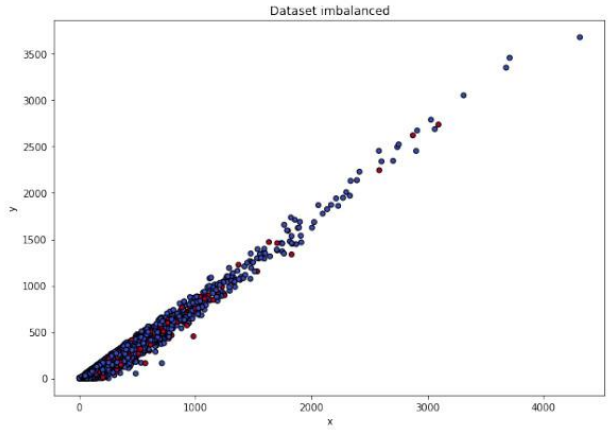
[12]:
```
clf_us = GaussianNB().fit(X_train_us,y_train_us)

predictions_us = clf_us.predict(X_test)
print(classification_report(y_test,predictions_us))
y_pred_proba_rf_us = clf_us.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo Naive Bayes utilizando UnderSampling: %.
 ↪5f' % clf_us.score(X_test,y_test))

print('\n\n\n')

clf_os = GaussianNB().fit(X_train_os,y_train_os)

predictions_os = clf_os.predict(X_test)
print(classification_report(y_test,predictions_os))
y_pred_proba_rf_os = clf_os.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo Naive Bayes utilizando OverSampling: %.
 ↪5f' % clf_os.score(X_test,y_test))

#ROC Curve
fpr_us, tpr_us, _ = roc_curve(y_test,y_pred_proba_rf_us)
roc_auc_us = auc(fpr_us, tpr_us)

fpr_os, tpr_os, _ = roc_curve(y_test,y_pred_proba_rf_os)
roc_auc_os = auc(fpr_os, tpr_os)
```

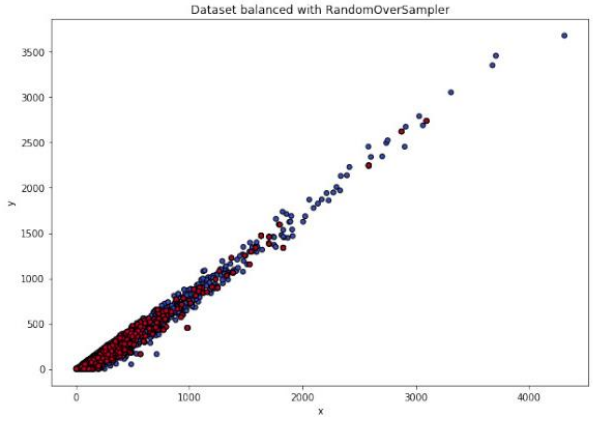
```
              precision    recall  f1-score   support

         0.0       0.95      0.25      0.39       372
         1.0       0.22      0.94      0.36        86
```

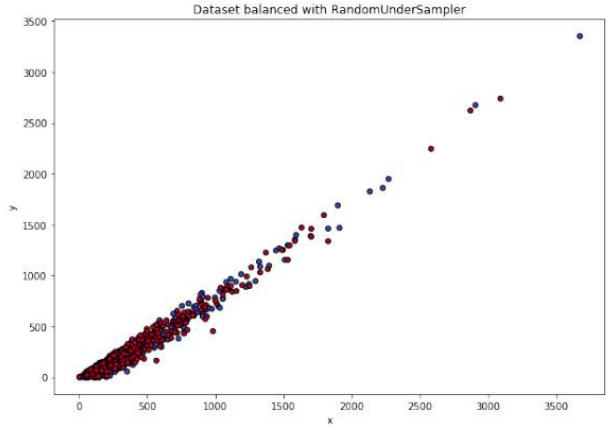
```
    accuracy                           0.38       458
   macro avg       0.59      0.59      0.38       458
weighted avg       0.81      0.38      0.39       458

Porcentaje de acierto del modelo Naive Bayes utilizando UnderSampling: 0.37773
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.15      0.26       372
         1.0       0.21      0.95      0.34        86

    accuracy                           0.30       458
   macro avg       0.57      0.55      0.30       458
weighted avg       0.80      0.30      0.27       458

Porcentaje de acierto del modelo Naive Bayes utilizando OverSampling: 0.29913
```

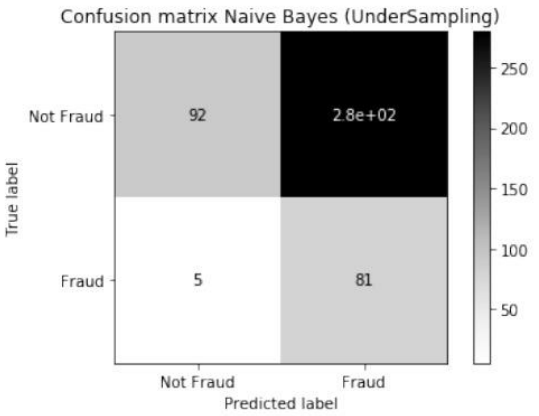
[13]:
```
plt.figure(figsize=(10,7))

plt.title('Dataset imbalanced')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(X_train[:, 0], X_train[:, 1], marker='o', c=y_train,
            s=25, edgecolor='k', cmap=plt.cm.coolwarm)
plt.show()

plt.figure(figsize=(10,7))
plt.title('Dataset balanced with RandomUnderSampler')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(X_train_us[:, 0], X_train_us[:, 1], marker='o', c=y_train_us,
            s=25, edgecolor='k', cmap=plt.cm.coolwarm)
plt.show()

plt.figure(figsize=(10,7))
plt.title('Dataset balanced with RandomOverSampler')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(X_train_os[:, 0], X_train_os[:, 1], marker='o', c=y_train_os,
            s=25, edgecolor='k', cmap=plt.cm.coolwarm)
plt.show()
```

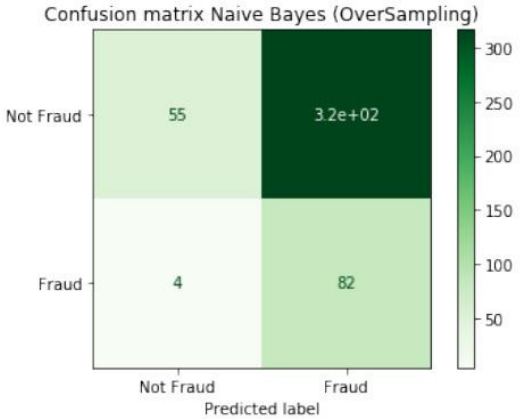






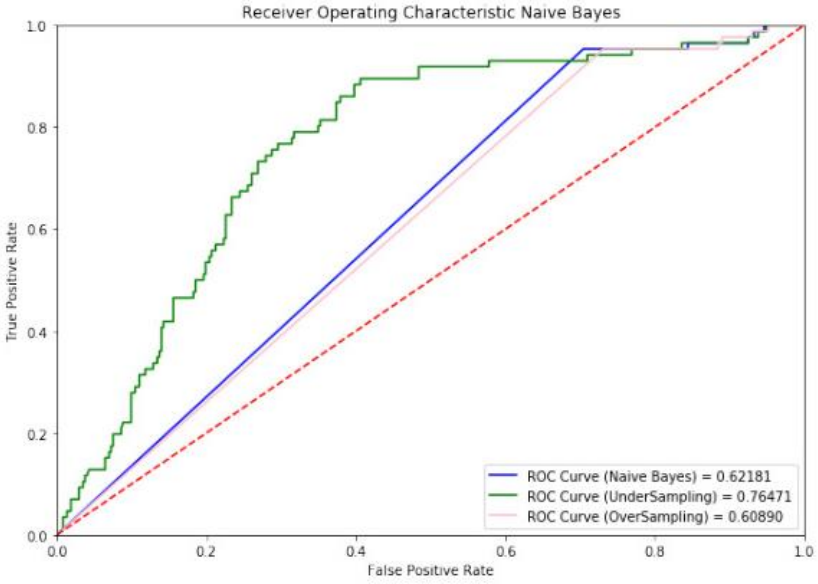
[14]:
```
disp_us = plot_confusion_matrix(clf_us, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Greys)
disp_us.ax_.set_title('Confusion matrix Naive Bayes (UnderSampling)')

disp_os = plot_confusion_matrix(clf_os, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Greens)
disp_os.ax_.set_title('Confusion matrix Naive Bayes (OverSampling)')

plt.show()
```





0.2.1 Curva ROC

[15]:
```
plt.figure(figsize=(10,7))
plt.title('Receiver Operating Characteristic Naive Bayes')

plt.plot(fpr_nb, tpr_nb, 'blue', label = 'ROC Curve (Naive Bayes) = %.5f' %␣
 ↪roc_auc_nb)
plt.plot(fpr_us, tpr_us, 'green', label = 'ROC Curve (UnderSampling) = %.5f' %␣
 ↪roc_auc_us)
plt.plot(fpr_os, tpr_os, 'pink', label = 'ROC Curve (OverSampling) = %.5f' %␣
 ↪roc_auc_os)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



0.3 Balanced Random Forest

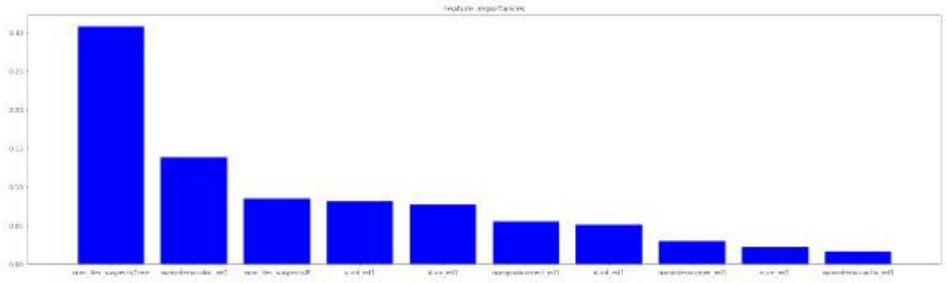
[16]:
```
clf_brf = BalancedRandomForestClassifier(n_estimators=10,
                                         max_depth=5).fit(X_train,y_train)

predictions_brf = clf_brf.predict(X_test)
print(classification_report(y_test,predictions_brf))

y_pred_proba_rf_brf = clf_brf.predict_proba(X_test)[:,1]
print('Porcentaje de acierto del modelo Random Forest: %.5f' % clf_brf.
 ↪score(X_test,y_test))

#ROC Curve
fpr_brf, tpr_brf, _ = roc_curve(y_test,y_pred_proba_rf_brf)
roc_auc_brf = auc(fpr_brf, tpr_brf)
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.83      0.89       372
         1.0       0.53      0.84      0.65        86

    accuracy                           0.83       458
   macro avg       0.74      0.83      0.77       458
weighted avg       0.88      0.83      0.84       458

Porcentaje de acierto del modelo Random Forest: 0.82969
```

[17]:
```
df = pd.read_csv("/home/blanca/Documentos/doc_TFG_actualizacion/
 ↪data_suspect_new_features/dataset_new_features.csv",sep=',')

data = df.values

X_a = data[:,:-1]
y_a = data[:,-1]

X_train_a, X_test_a, y_train_a, y_test_a = train_test_split(X_a, y_a,␣
 ↪test_size=0.15)

clf_abrf = BalancedRandomForestClassifier(n_estimators=10,
                                          max_depth=5).fit(X_train_a,y_train_a)

predictions_abrf = clf_abrf.predict(X_test_a)
print(classification_report(y_test_a,predictions_abrf))

y_pred_proba_rf_abrf = clf_abrf.predict_proba(X_test_a)[:,1]
print('Porcentaje de acierto del modelo Random Forest Agregrando ' +
      'nuevos atributos al dataset: %.5f' % clf_abrf.score(X_test_a,y_test_a))

#ROC Curve
fpr_abrf, tpr_abrf, _ = roc_curve(y_test_a,y_pred_proba_rf_abrf)
roc_auc_abrf = auc(fpr_abrf, tpr_abrf)
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.95      0.96       377
         1.0       0.80      0.88      0.84        81

    accuracy                           0.94       458
   macro avg       0.89      0.91      0.90       458
weighted avg       0.94      0.94      0.94       458

Porcentaje de acierto del modelo Random Forest Agregrando nuevos atributos al
dataset: 0.93886
```

[18]:
```
# Calculamos los atributos más relevantes para el modelo
importances = clf_abrf.feature_importances_
# Los ordenamos de manera descendente y nos quedamos con los 10 más importantes
indices = np.argsort(importances)[::-1][:10]

# Obtenemos el nombre de los atributos para mostrarlos en la siguiente figura
names = df.columns.values

plt.figure(figsize=(25,7))
plt.title("Feature importances")
plt.bar(range(indices.shape[-1]), importances[indices],
       color="b", align="center")
plt.xticks(range(indices.shape[-1]), names[indices])
plt.xlim([-1,indices.shape[-1]])

plt.show()
```



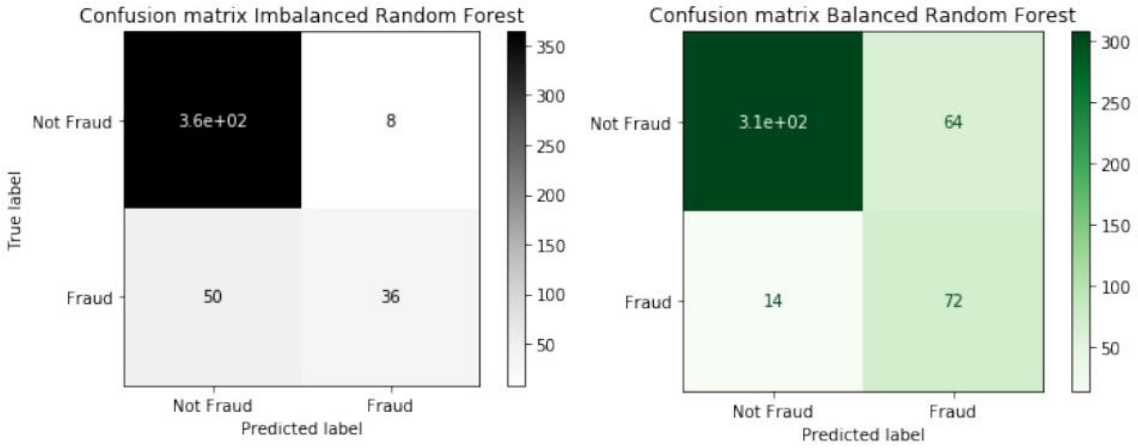
0.3.1 Balanced Random Forest - Drop features

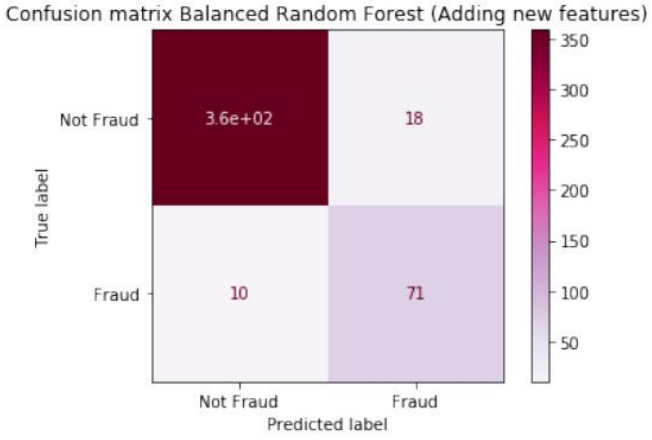
[19]:
```
X_d = data[:,indices]
y_d = data[:,-1]

X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_d, y_d,␣
 ↪test_size=0.15)

clf_dbrf = BalancedRandomForestClassifier(n_estimators=100,
                                          max_depth=5).fit(X_train_d,y_train_d)

predictions_dbrf = clf_dbrf.predict(X_test_d)
print(classification_report(y_test_d,predictions_dbrf))

y_pred_proba_rf_dbrf = clf_dbrf.predict_proba(X_test_d)[:,1]
print('Porcentaje de acierto del modelo Random Forest Agregrando ' +
      'nuevos atributos al dataset: %.5f' % clf_dbrf.score(X_test_d,y_test_d))

#ROC Curve
fpr_dbrf, tpr_dbrf, _ = roc_curve(y_test_d,y_pred_proba_rf_dbrf)
roc_auc_dbrf = auc(fpr_dbrf, tpr_dbrf)
```

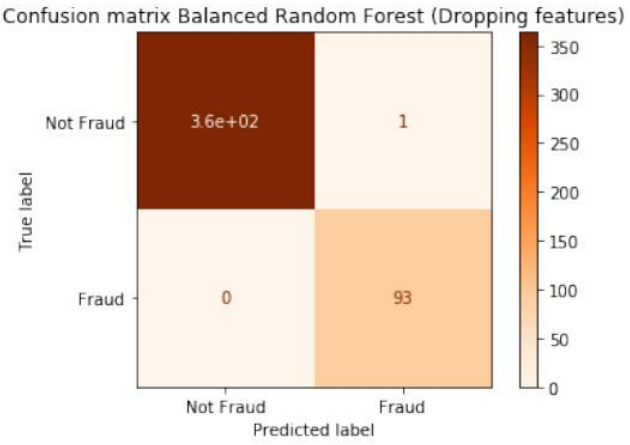
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       365
         1.0       0.99      1.00      0.99        93

    accuracy                           1.00       458
   macro avg       0.99      1.00      1.00       458
weighted avg       1.00      1.00      1.00       458

Porcentaje de acierto del modelo Random Forest Agregrando nuevos atributos al
dataset: 0.99782
```

0.3.2 Matriz de confusión

[20]:
```
disp_rf = plot_confusion_matrix(clf_rf, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Greys)
disp_rf.ax_.set_title('Confusion matrix Imbalanced Random Forest')

disp_brf = plot_confusion_matrix(clf_brf, X_test, y_test,
                                display_labels=target_names,
                                cmap=plt.cm.Greens)
disp_brf.ax_.set_title('Confusion matrix Balanced Random Forest')
disp_abrf = plot_confusion_matrix(clf_abrf, X_test_a, y_test_a,
                                display_labels=target_names,
                                cmap=plt.cm.PuRd)
disp_abrf.ax_.set_title('Confusion matrix Balanced Random Forest (Adding new␣
↪features)')

disp_dbrf = plot_confusion_matrix(clf_dbrf, X_test_d, y_test_d,
                                display_labels=target_names,
                                cmap=plt.cm.Oranges)
disp_dbrf.ax_.set_title('Confusion matrix Balanced Random Forest (Dropping␣
↪features)')

plt.show()
```







0.3.3 Curva ROC

[21]:
```
plt.figure(figsize=(10,7))
plt.title('Receiver Operating Characteristic Random Forest')

plt.plot(fpr_rf, tpr_rf, 'grey', label = 'ROC Curve (Imbalanced Random Forest)␣
↪= %.5f' % roc_auc_rf)
plt.plot(fpr_brf, tpr_brf, 'green', label = 'ROC Curve (Balanced Random Forest)␣
↪= %.5f' % roc_auc_brf)
plt.plot(fpr_abrf, tpr_abrf, 'blue', label = 'ROC Curve (Balanced Random Forest␣
↪+ Adding new features) = %.5f' % roc_auc_abrf)
plt.plot(fpr_dbrf, tpr_dbrf, 'orange', label = 'ROC Curve (Balanced Random␣
↪Forest + Dropping features) = %.5f' % roc_auc_dbrf)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```